US005920893A

United States Patent [19]

Nakayama et al.

[11] **Patent Number: 5,920,893**

[45] **Date of Patent: Jul. 6, 1999**

[54] **STORAGE CONTROL AND COMPUTER SYSTEM USING THE SAME**

[75] Inventors: **Shinichi Nakayama**, Odawara; **Shizuo Yokohata**, Kanagawa-ken, both of Japan

[73] Assignee: **Hitachi, Ltd.**, Tokyo, Japan

[21] Appl. No.: **08/867,191**

[22] Filed: **Jun. 2, 1997**

[30] **Foreign Application Priority Data**

Jun. 4, 1996 [JP] Japan 8-163927

[51] **Int. Cl.**^{6} **G06F 12/00**

[52] **U.S. Cl.** **711/147**; 711/147; 711/163; 711/170; 711/206; 395/494; 395/878; 395/200.08; 360/48; 360/369

[58] **Field of Search** 711/147, 163, 711/170, 206; 395/494, 878, 200.08; 360/48, 369

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,206,939	4/1993	Yanai et al.	395/400
5,581,743	12/1996	Burton et al.	395/500
5,664,144	9/1997	Yanai et al.	711/113
5,724,542	3/1998	Taroda et al.	711/113
5,778,349	7/1998	Okonogi	707/1

FOREIGN PATENT DOCUMENTS

60-254270	12/1985	Japan .
63-211019	9/1988	Japan .
1-309117	12/1989	Japan .
03256945	11/1991	Japan .

Primary Examiner—Tod R. Swann
Assistant Examiner—Nasser Moazzami
Attorney, Agent, or Firm—Antonelli, Terry, Stout & Kraus, LLP

[57] **ABSTRACT**

A storage control enables data on various storage media to be shared among host computers having various different host computer input/output interfaces. A control processor checks a host computer interface management table when write is requested by a host computer (HCP). The control processor writes write data in a cache slot of the cache memory without converting the format if the data format of the HCP is in an FBA format and it converts the format into the FBA format and writes it when the data format of HCP is in a CKD format. The processor checks a table when read is requested by HCP. If the data format of HCP is FBA, the processor transfers the read data read from the cache slot without converting it and if it is CKD, the processor converts the format into the FBA format and transfers the format converted data. The control processors, retrieve write data in the cache memory and writes it in a drive.

19 Claims, 9 Drawing Sheets

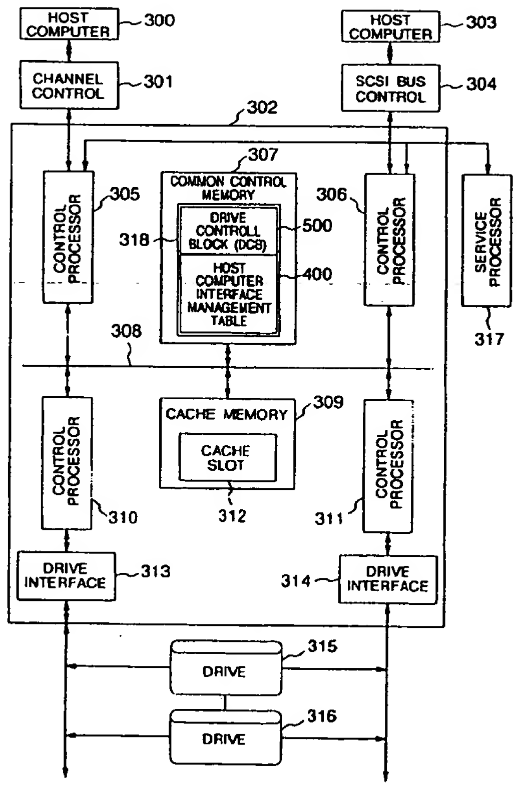


FIG. 1

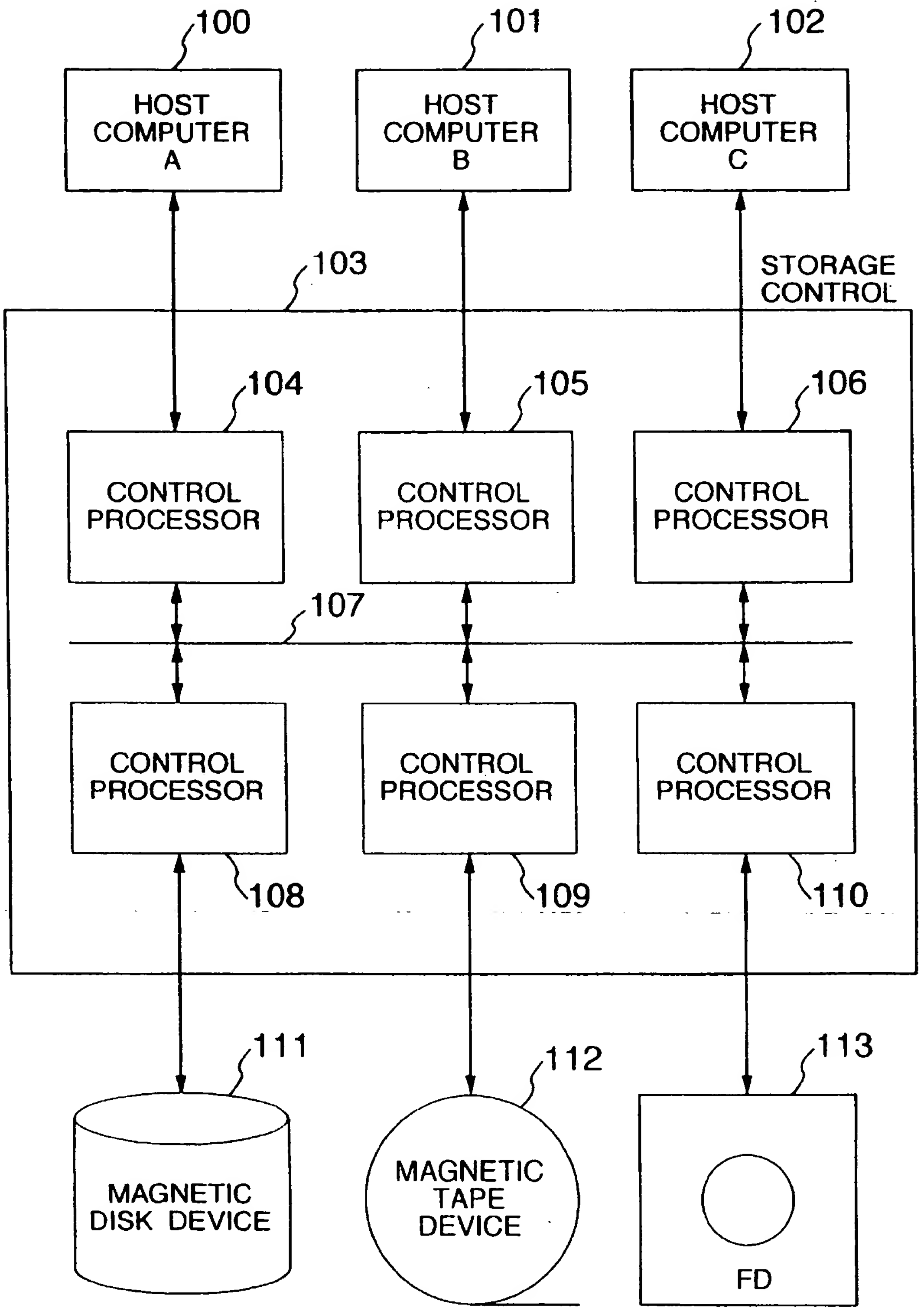

100
HOST COMPUTER A
101
HOST COMPUTER B
102
HOST COMPUTER C
103
STORAGE CONTROL
104
CONTROL PROCESSOR
105
CONTROL PROCESSOR
106
CONTROL PROCESSOR
107
CONTROL PROCESSOR
CONTROL PROCESSOR
CONTROL PROCESSOR
108
109
110
111
MAGNETIC DISK DEVICE
112
MAGNETIC TAPE DEVICE
113
FD

FIG. 2

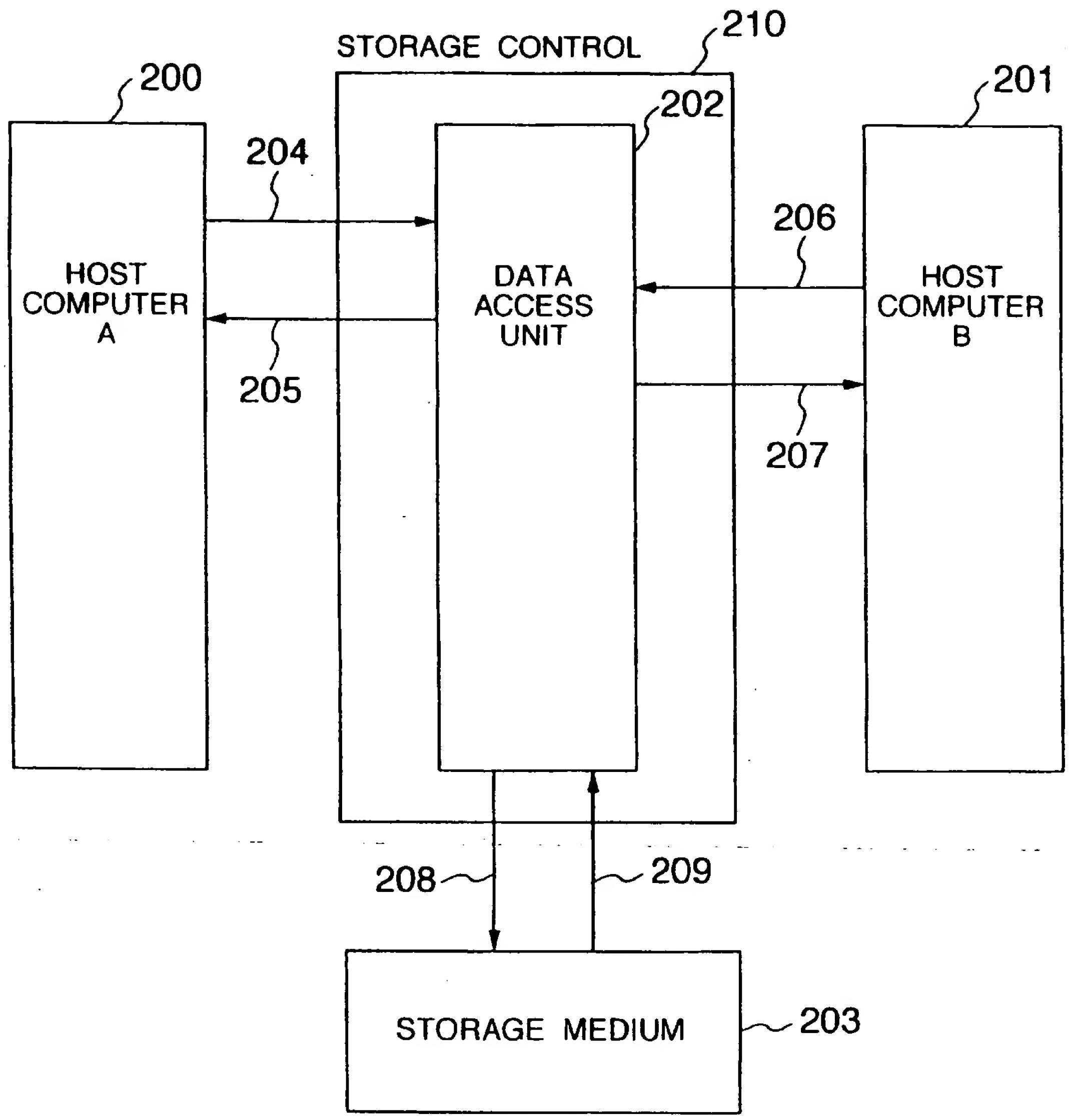

STORAGE CONTROL
210
200
202
201
204
HOST COMPUTER A
DATA ACCESS UNIT
206
HOST COMPUTER B
205
207
208
209
STORAGE MEDIUM
203

FIG. 3

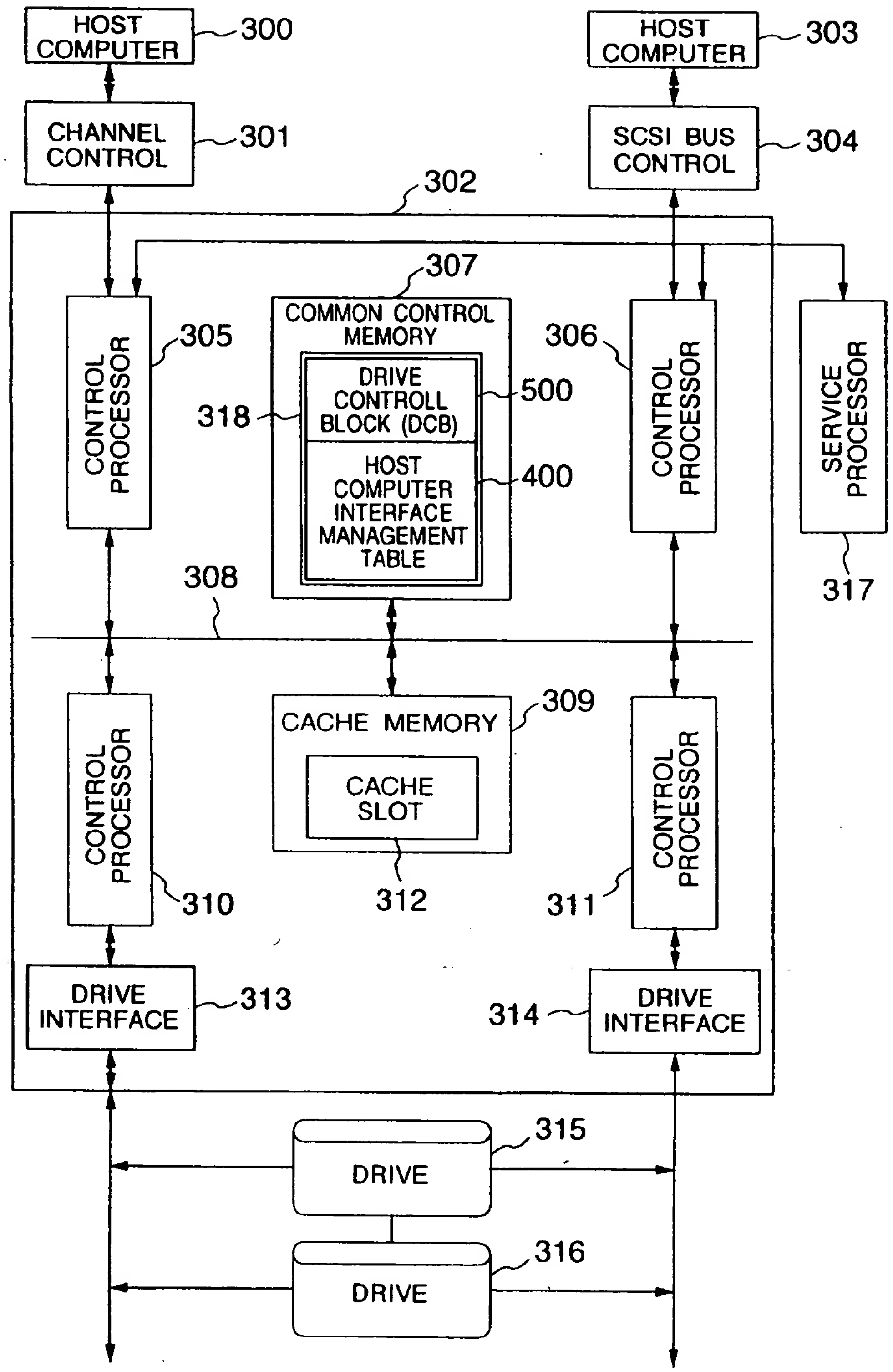

HOST COMPUTER
300
HOST COMPUTER
303
CHANNEL CONTROL
301
SCSI BUS CONTROL
304
302
307
COMMON CONTROL MEMORY
CONTROL PROCESSOR
305
DRIVE CONTROLL BLOCK (DCB)
500
318
HOST COMPUTER INTERFACE MANAGEMENT TABLE
400
306
CONTROL PROCESSOR
SERVICE PROCESSOR
317
308
CACHE MEMORY
309
CACHE SLOT
312
CONTROL PROCESSOR
310
CONTROL PROCESSOR
311
DRIVE INTERFACE
313
314
DRIVE INTERFACE
DRIVE
315
DRIVE
316

FIG. 4

DRIVE CONTROL BLOCK (DCB)

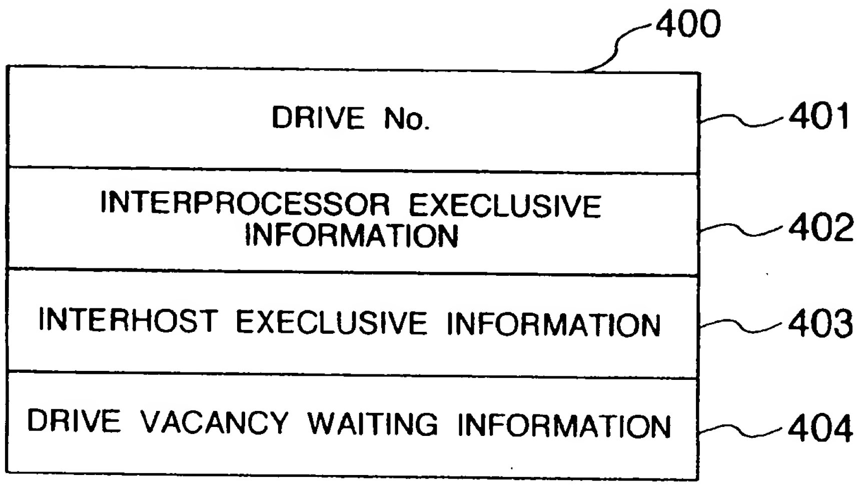
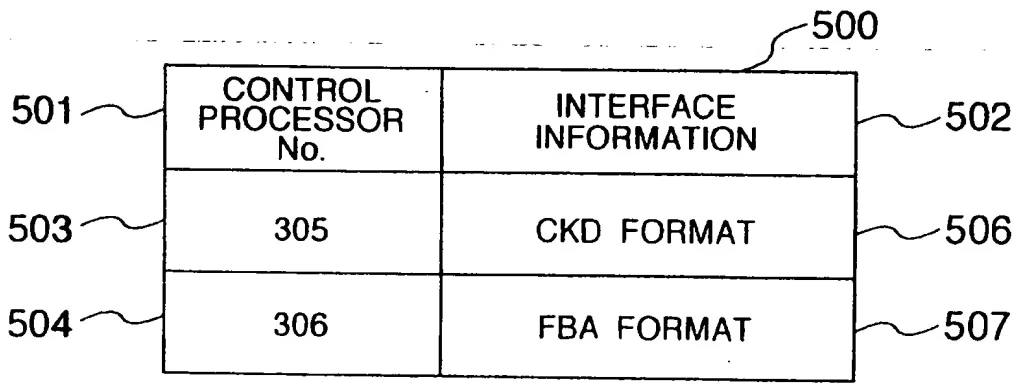

400
DRIVE No.
401
INTERPROCESSOR EXECLUSIVE INFORMATION
402
INTERHOST EXECLUSIVE INFORMATION
403
DRIVE VACANCY WAITING INFORMATION
404

FIG. 5

HOST COMPUTER INTERFACE MANAGEMENT INFORMATION TABLE


500
501
502
503
504
506
507

CONTROL PROCESSOR No.	INTERFACE INFORMATION
305	CKD FORMAT
306	FBA FORMAT

FIG.6

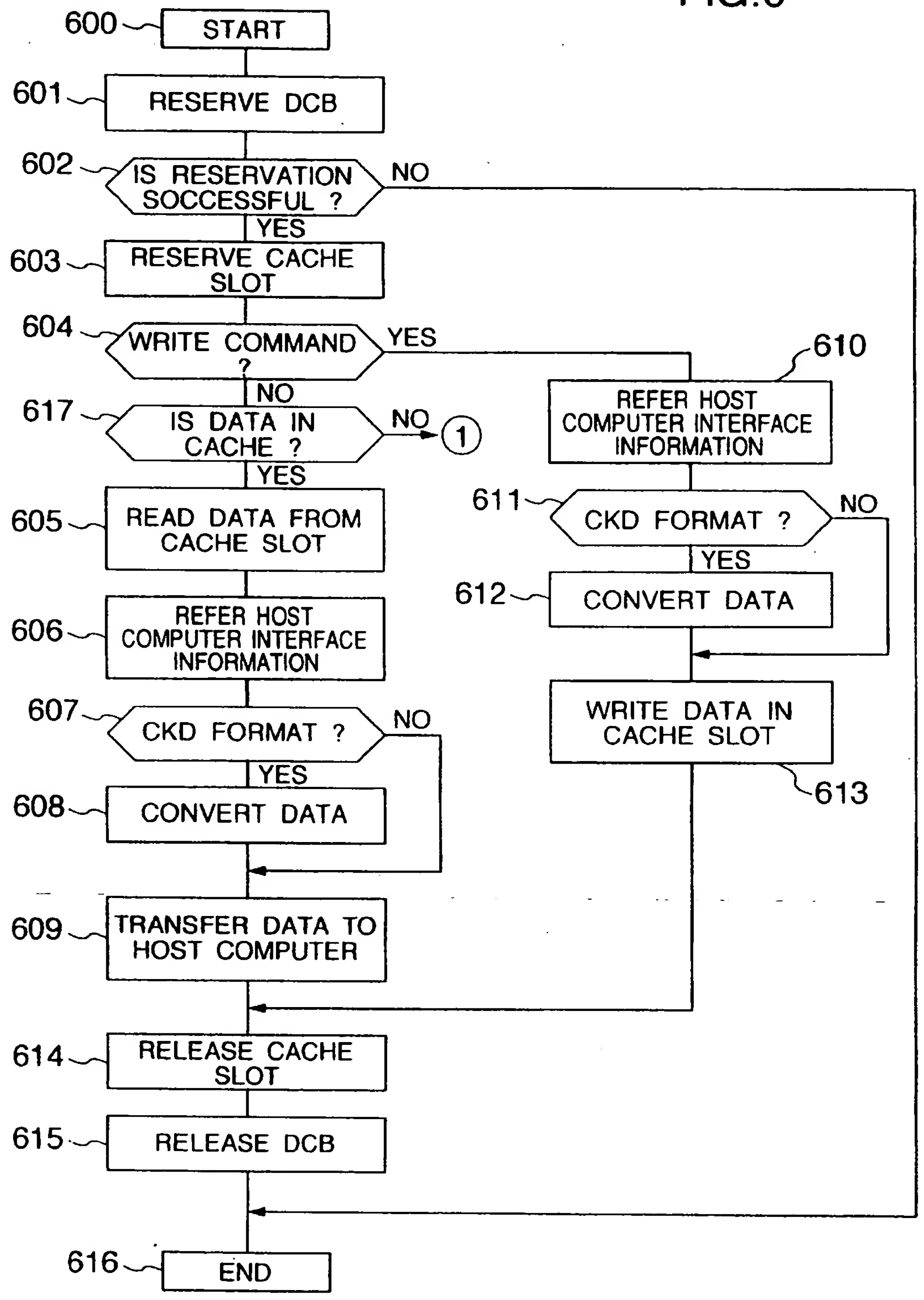

600 START
601 RESERVE DCB
602 IS RESERVATION SOCCESSFUL ?
NO
YES
603 RESERVE CACHE SLOT
604 WRITE COMMAND ?
YES
NO
617 IS DATA IN CACHE ?
NO
1
YES
605 READ DATA FROM CACHE SLOT
606 REFER HOST COMPUTER INTERFACE INFORMATION
607 CKD FORMAT ?
NO
YES
608 CONVERT DATA
609 TRANSFER DATA TO HOST COMPUTER
610 REFER HOST COMPUTER INTERFACE INFORMATION
611 CKD FORMAT ?
NO
YES
612 CONVERT DATA
WRITE DATA IN CACHE SLOT 613
614 RELEASE CACHE SLOT
615 RELEASE DCB
616 END

FIG.7

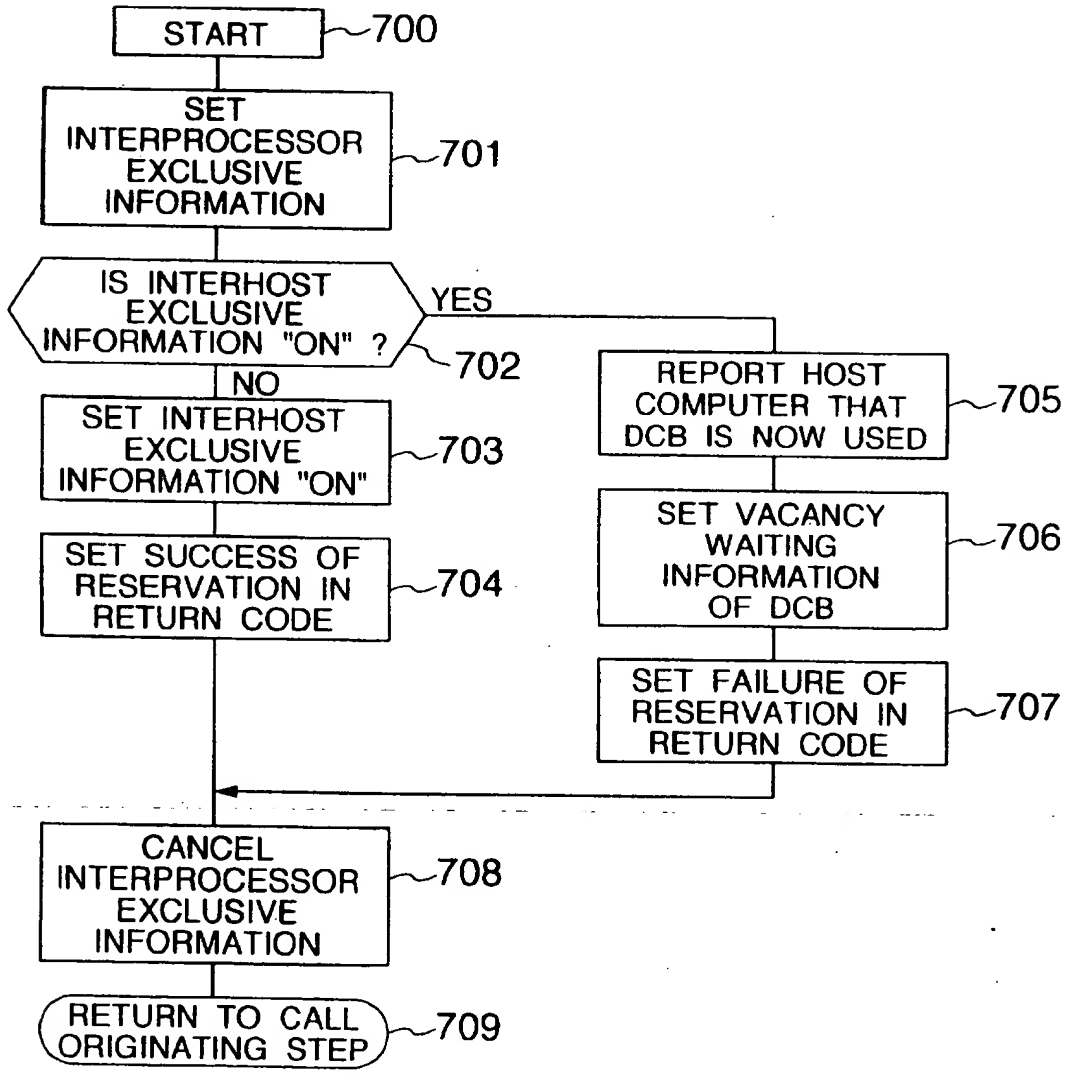

START
700
SET INTERPROCESSOR EXCLUSIVE INFORMATION
701
IS INTERHOST EXCLUSIVE INFORMATION "ON" ?
702
YES
NO
SET INTERHOST EXCLUSIVE INFORMATION "ON"
703
SET SUCCESS OF RESERVATION IN RETURN CODE
704
REPORT HOST COMPUTER THAT DCB IS NOW USED
705
SET VACANCY WAITING INFORMATION OF DCB
706
SET FAILURE OF RESERVATION IN RETURN CODE
707
CANCEL INTERPROCESSOR EXCLUSIVE INFORMATION
708
RETURN TO CALL ORIGINATING STEP
709

FIG.8

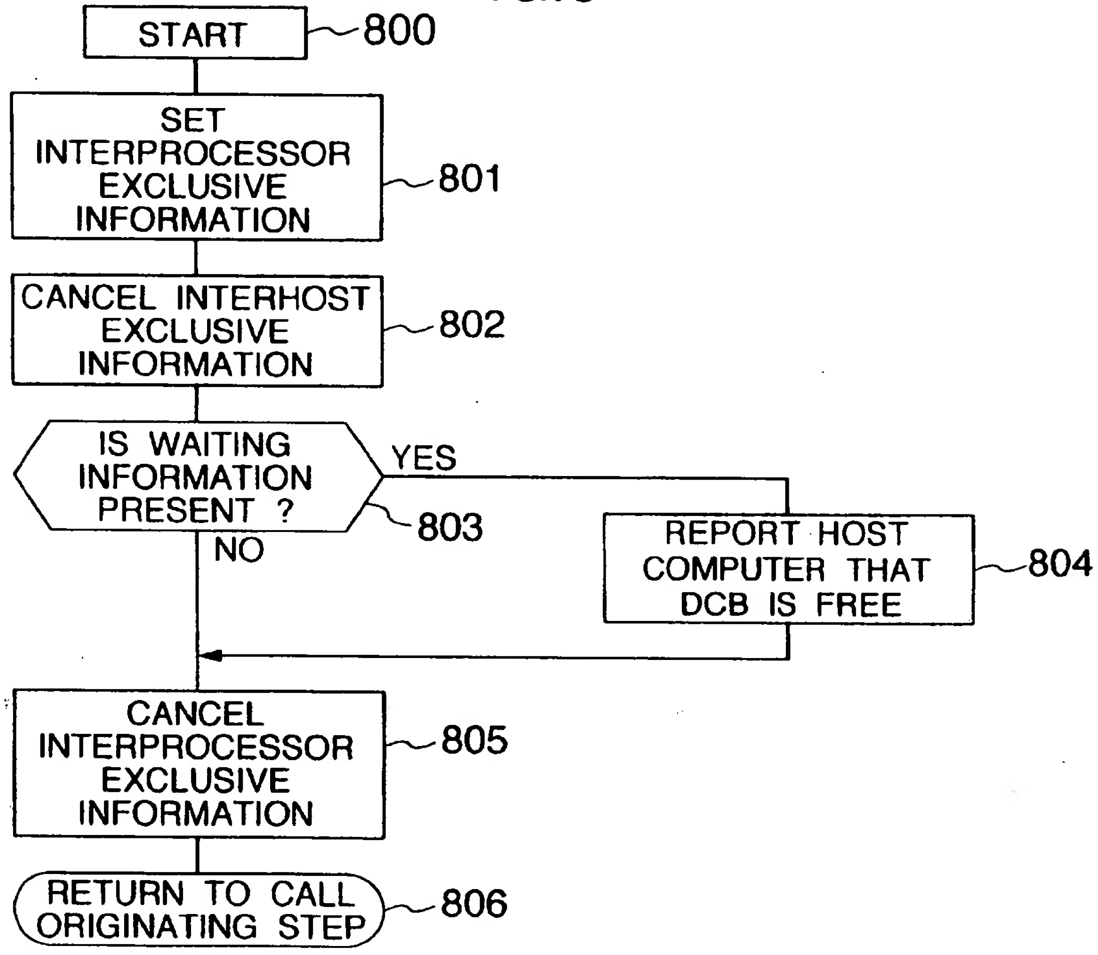

START
800
SET INTERPROCESSOR EXCLUSIVE INFORMATION
801
CANCEL INTERHOST EXCLUSIVE INFORMATION
802
IS WAITING INFORMATION PRESENT ?
YES
NO
803
REPORT HOST COMPUTER THAT DCB IS FREE
804
CANCEL INTERPROCESSOR EXCLUSIVE INFORMATION
805
RETURN TO CALL ORIGINATING STEP
806

FIG. 9

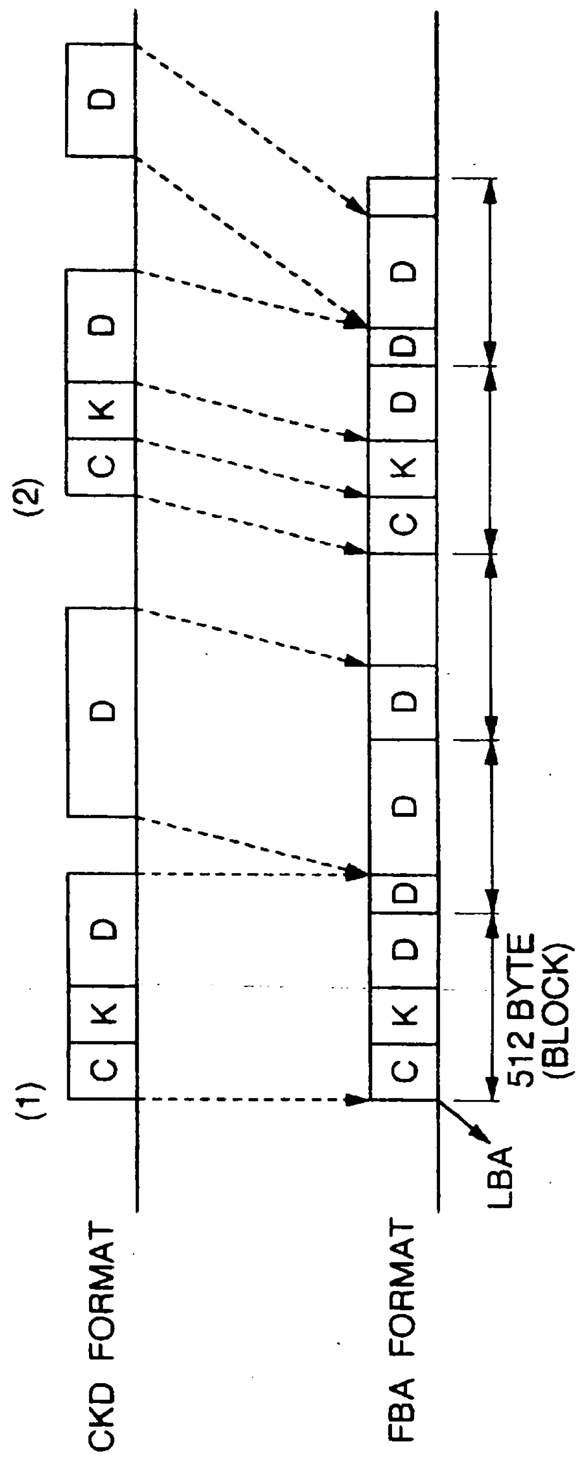
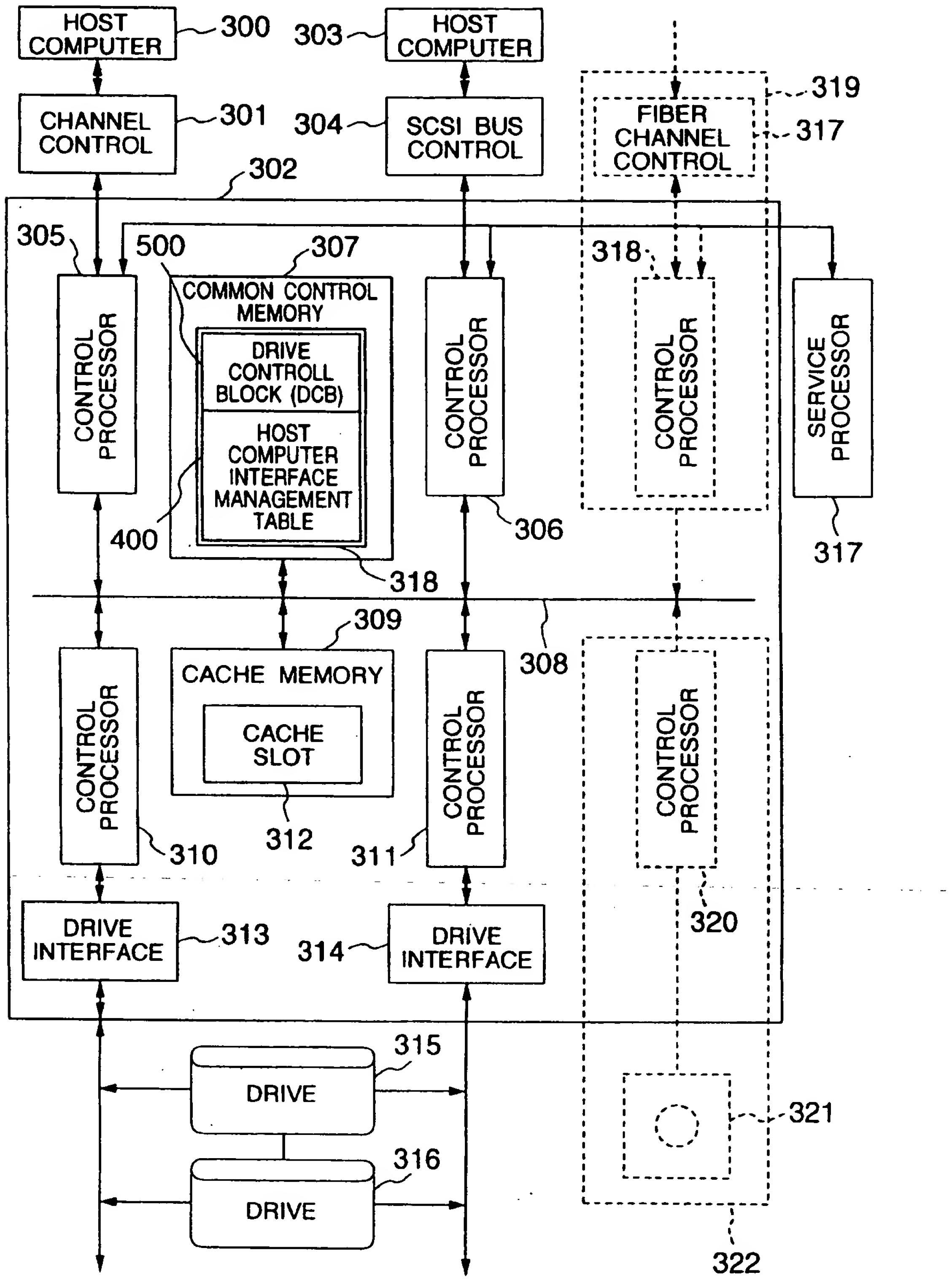


FIG. 10


HOST COMPUTER
300
303
HOST COMPUTER
CHANNEL CONTROL
301
304
SCSI BUS CONTROL
319
FIBER CHANNEL CONTROL
317
302
305
500
307
COMMON CONTROL MEMORY
DRIVE CONTROLL BLOCK (DCB)
HOST COMPUTER INTERFACE MANAGEMENT TABLE
CONTROL PROCESSOR
CONTROL PROCESSOR
318
CONTROL PROCESSOR
SERVICE PROCESSOR
400
306
317
318
309
308
CACHE MEMORY
CACHE SLOT
CONTROL PROCESSOR
CONTROL PROCESSOR
CONTROL PROCESSOR
310
312
311
320
DRIVE INTERFACE
313
314
DRIVE INTERFACE
315
DRIVE
316
DRIVE
321
322

STORAGE CONTROL AND COMPUTER SYSTEM USING THE SAME

BACKGROUND OF THE INVENTION

The present invention relates to a storage control and in particular to a storage control system which enables data on various storage media for storing input/output data to/from host computers to be shared among said host computers having various host computer input/output interfaces; and a computer system using the same.

Recently cases have been increased in which main frames are linked with a division system of an open system bases, such as downsizing of part of operations (transactions, jobs) which used to be processed by a main frame to a division server (for example, UNIX server, etc.) or incorporation of an information system into a division.

In these cases, due to the fact that the data format (CKD format) of the main frame is different from the data format (FBA format) of the host computer input/output interface of the UNIX server, development of programs for data conversion and data conversion between host computers is required or a storage control devoted to each host computer input/output interface is necessary. This makes it difficult to build a wide range of computer system configurations.

As one of methods which have been devised in order to overcome the above-mentioned problems, an integrated computer system which enables various programs to be executed without limiting the CPU (central processing unit) architecture by adopting, for example, a hardware configuration including a plurality of computers in one system is disclosed in JP-A-60-254270.

A magnetic disk device including an interface which is compatible to a plurality of different interface standards and an interface conversion control circuit which enables files on a magnetic disk to be shared is disclosed in JP-A-1-309117.

SUMMARY OF THE INVENTION

In the prior art technology which is disclosed in the above-mentioned JP-A-60-254270, CPUs having different architectures have a master-slave relationship with each other. The CPU on the other slave side having different architecture is exclusively prevented to simultaneously use the storage medium in order that the CPU on the slave side is selected by a hardware switch to occupy a system bus for executing an input/output operation to/from the storage medium. Accordingly when the selected CPU is used for an extended period of time, a disadvantage occurs that the storage medium and/or the system bus which are resources common to the system would be occupied for an extended period of time.

Since files in the magnetic disk device are divided and stored for each of CPU sharing different architectures, it is impossible to share the same file in the magnetic disk device among CPUs having different architectures.

Although it is possible for host computers having different architectures to share a storage medium in the prior art as mentioned above, the storage medium is exclusively used among host computers having different architectures. This may partly cause the utilization efficiency of the file subsystem to be remarkably lowered. The disadvantage that data sharing among host computers having different host computer input/output interfaces is not overcome.

Although file sharing among host devices having different interfaces is possible in the above-mentioned JP-A-1-309117, data sharing among different interfaces is not mentioned.

It is an object of the present invention to provide a storage control in which a request of data access to a storage medium from host computers having different host computer input/output interfaces is made possible by conducting data conversion if it is necessary for such a request and in which sharing of data on the storage medium among host computers having different host computer input/output interfaces is made possible whereby extensibility of file subsystem and responsiveness of data is enhanced to enable a wide rage of computer system configurations to be built.

It is another object of the present invention to provide a storage control which enables various host computer input/output interfaces and/or various storage medium input/output interfaces to be added to or removed from the storage control.

It is a further object of the present invention to provide a computer system including the above-mentioned storage control.

In order to accomplish the above-mentioned object, in an aspect of the present invention there is provided a storage control in a computer system including a plurality of host computers having various different host computer input/output interfaces, the storage control for controlling input/output to/from the host computers, and various storage media for storing input/output data of the host computers,

wherein the storage control is made up of: control processors, each being connected to one of host computers; device input/output interfaces, each for one of storage media, for connecting the control processors with the various storage media to input/output data in a predetermined format to/from each storage medium; and a host interface management table for managing the data format of the host computer interface of each host computer;

each of the control processors includes a data format converting unit which compares the data format of corresponding host computer in the host computer interface managing table with the predetermined data format when a write data request is issued from the corresponding host computer, convertes the data format of the write data into the predetermined data format, writes the converted write data in the storage medium when the compared formats match with each other and writes the write data without converting the data format of the write data when they do not match with each other, and which also compares the data format of the corresponding host computer in the host computer interface managing table with the predetermined data format when a read data request is issued from the corresponding host computer, converts the data format of the read data read from the storage medium into the predetermined data format to transfer the converted read data to the corresponding host computer when the compared formats do not match with each other and transfers the read data to the corresponding host computer without converting the data format of the read data when they match with each other. This configuration enables data on various storage media to be shared among host computers having different host computer input/output interfaces.

In a computer system having the storage control of the above-mentioned configuration, addition or removal of one or more host computers having desired kinds of host computer input/output interfaces and one or more control processors compatible to the host computers is made possible by updating the host computer interface managing table.

In accordance with one feature of the present invention, drive control blocks (DCB), each for one of storage devices (drives) are provided for controlling the access to each storage device. This enables various storage devices having a given device input/output interface to be added or removed.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic block diagram showing one embodiment of a computer system of the present invention;

FIG. 2 is a schematic block diagram which is useful for explaining the summary of the present invention;

FIG. 3 is a schematic block diagram showing the configuration of another embodiment of a disk subsystem of the present invention;

FIG. 4 is a diagram showing the configuration of a drive control block;

FIG. 5 is a diagram showing one example of host computer interface management information;

FIG. 6 is a flow chart showing a data access operation;

FIG. 7 is a flow chart showing a DCB reserving operation;

FIG. 8 is a flow chart showing a DCB releasing operation;

FIG. 9 is a chart useful for illustrating the conversion between CKD and BAD formats; and

FIG. 10 is a schematic block diagram showing the configuration of a further embodiment of a disk subsystem of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Now, the embodiments of the present invention will be described in detail with reference to drawings.

FIG. 1 is a block diagram which is useful for explaining the principle of the operation of the present invention and showing the configuration of an embodiment of a computer system of the present invention which comprises a plurality of host computers having various different input/outputs therefor, a storage control for controlling the input/output to/from the host computers and various storage media for storing therein input/output data to/from the host computers.

In FIG. 1, the plurality of host computers A100, B101 and C102 having various different host computer input/output interfaces are connected to a magnetic disc device 111, magnetic tape device 112 and floppy disk device 113 via a storage control 103.

Control processors 104, 105, 106, 108, 109 and 110 which are incorporated in the storage control 103 are adapted to control the transfer of data among the host computers A100, B101 and C102 and the magnetic disk device 111, magnetic tape device 112 and the floppy disk device 113.

In other words, the control processors 104, 105 and 106 execute an input/output data transfer request from the host computers A100, B101 and C102 and the control processors 108, 109 and 110 execute a request of input/output data transfer to the magnetic disk device 111, magnetic tape device 112 and the floppy disk device 113.

All the control processors 104, 105, 106, 108, 109 and 110 receive and transmit data and control signal with each other via a control line 107.

FIG. 2 is a block diagram for explaining the outline of the present invention. Now, the outline of the present invention will be described with reference to FIG. 2.

In FIG. 2, the host computers A200, B201 having different host computer input/output interfaces and the storage

medium 203 are connected to the storage control 210, which includes a data access unit 202. An access request 204 (write), 205 (read), 206 (write) and 207 (read) which are generated by the host computers A200 and B201 to the storage media 203 are issued to the storage control 210. These access requests are executed by the data access unit 202.

If the write access request 204 from the host computer A200 is issued, the data access unit 202 performs data conversion when the data conversion of the write data is necessary and writes converted data on the storage medium 203 and write unconverted write data on the storage medium 203 when the data conversion is not necessary.

If the write access data request 204 is issued from the host computer A200, the data access unit 202 performs data conversion of the write data when it is necessary and writes the converted data on storage medium 203 and writes unconverted write data on the storage medium 203 when the data conversion is not necessary (208).

If the read access request 205 from the host computer A200 is issued, the data access unit 202 reads data from the storage medium and performs data conversion when it is necessary and transfers the converted data to the host computer A and transfers the unconverted read data to the host computer A when the data conversion is not necessary.

Processing for the access request from the host computer B201 is conducted similarly to the processing for the access request from the host computer A200.

The above-mentioned operation enables the host computers having various different host computer input/output interfaces to share the data on the storage medium.

It is to be noted that the number of the host computers having different host computer input/output interfaces is not limited to 2, the three or more host computers may be connected.

FIG. 3 is a block diagram showing the configuration of a disk subsystem having a cache memory in another embodiment of the present invention.

In FIG. 3, a disk control 302 is connected to a host computer 300 via a channel control 301 on the host side and is also connected to a host computer 303 via a small computer system interface (abbreviated as SCSI).

In the present embodiment, the host computer 300 is a main frame computer (CKD data format) and the host computer 303 is an UNIX computer (FBA data format).

The disk control 302 is connected to drives 315 and 316 which are magnetic storage medium on the lower side.

The disk control 302 performs read/write of data on the drives 315 and 316 in response to the requests of the host computers 300 and 303.

Data transfer among the host computers 300, 303 and the drives 315, 316 is controlled by the control processors 305, 306, 310 and 311 which are incorporated in the disk control 302.

The control processors 305 and 306 are connected to the host computers 300 and 303 through the channel control 301 and the SCSI bus control 304, respectively and the control processors 310 and 311 are connected to the drives 315 and 316 through the drive interfaces 313 and 314, respectively.

The control processors 305 and 306 mainly perform data transfer between the host computers 300 and 303 and the cache memory 309. The control processors 310 and 311 mainly perform data transfer between the cache memory 309 and the drives 315 and 316.

A common control memory 307 is a common memory which is accessible from all control processors 305, 306, 310 and 311 and stores therein common control information 318 used for allowing the disk control 302 to manage the drives 315 and 316. The common control information 318 will be described hereafter in detail.

The cache memory 309 is accessible from all control processors 305, 306, 310 and 311 and is used for temporarily storing data which is read from the drives 315 and 316. A cache slot 312 is the data management unit quantity in the cache memory 309.

The control processors 305, 306, 310 and 311 receive/transmit data and control signals from/to the cache memory 309 and the common control memory 307 via a signal line 308.

The control processors 305 and 306 are connected to a service processor 317.

When update of the common control information 318 in the common control memory 307 is instructed from the service processor 317, the service processor 317 selects any one of the control processors 305 and 306 to send an update request, so that the selected control processor will update the common control information 318 in the common control memory 307.

Now, the common control information will be described.

The common control information 318 includes a drive control block 400 and host computer interface management information table 500, which will be described in order.

FIG. 4 shows a drive control block (abbreviated as DCB) 400. One DCB 400 is provided for each one of the drives and stores therein four data.

The four data include a drive number 401 for enabling the disk control 302 to identify each drive, interprocessor exclusive data 402, interhost exclusive information 403 and drive vacancy waiting information 404.

The interprocessor exclusive information 402 is used when the control processor 305 or 306 exclusively controls the DCB access from the other control processor and sets the processor number when the control processor reserves the access right for the DCB having the specified drive number and cancels the processor number when the control processor releases the access right to the DCB.

The interhost exclusive information 403 is used when the host computer 300 or 303 exclusively controls the drive access from the other host computer and sets the information 403 "on" when the processor reserves access of the specified drive number and sets it "off" when the host computer releases the access right.

The drive vacancy waiting information 404 is used to inform one of the host computers 300 and 304 that a DCB having the drive number specified is vacant upon being released from use by the other host computer which has been using the DCB when the one computer requested to reserve the access right for the DCB but was informed of the fact that the DCB was being used by the other host computer.

FIG. 5 shows a host interface management information table 500.

In the table 500, the host interface information 502 is managed for each number 501 of the control processor of a host computer connected. It is determined based upon this information as to whether data conversion is to be conducted.

In the present embodiment, the control processors 305 (503 in FIG. 5) and 306 (504 in FIG. 5) manage the CKD and FBA formats 506 and 507, respectively. The present management information table 500 is set and reset in response to an instruction from the service processor 317.

Now, operation of the control processors 305, 306, 310 and 311 in the disk control 302 in accordance with the present invention will be described.

FIG. 6 is a flow chart showing the main flow of operation which is executed by a data access processing unit (600) including a data format conversion unit.

When a control processor 305 receives a data access command from the host computer 300, DCB processing for reserving the access right for the DCB of the specified drive number is executed (step 601).

A determination as to whether reservation of DCB is succeeded or not is made (step 602). If failed, the data access processing is ended (step 616). If it is successful, following operation will be conducted.

Firstly, reservation of the cache slot is conducted (step 603). Subsequently, a determination is made as to whether or not the data access command is a write command or a read command (step 604).

If it is a write command, operation will be conducted as follows:

With reference to the host computer interface management information table 500 (FIG. 5) (step 610), a determination is made as to whether the host computer interface which is connected to the control processor is in the CKD or FBA format (step 611).

In the present embodiment, the control processors 305 and 306 are in the CKD and FBA formats 506 and 507, respectively.

If it is in the CKD format, the control processor 305 converts CKD data into FBA data (step 612) and thereafter writes the converted write data into the cache slot 312 (step 613). If it is in the FBA format, the control processor 305 writes FBA data into the cache slot 312 without conducting any conversion. Conversion of FBA data into CKD data will be described hereafter Thereafter, the cache slot is released (step 614) and DCB is released (step 615).

If the received command is a read command, operation will be executed as follows:

Firstly, a determination is made as to whether a read data exists in the cache memory (step 617). If the data exists, operation at step 605 and the subsequent steps will be conducted. If no data exists, operation (1) will be conducted.

In the operation (1), the control processors 305 and 306 instruct the control processors 310, 311 to read data from the drives and the control processors 310, 311 read data from the drives via the drive interfaces 313, 314 to write read data in the cache slot of the cache memory. This operation is omitted in the flow chart of FIG. 6.

Now, operation at step 605 and the subsequent steps will be described.

Data on the cache slot 312 is read (step 605).

With reference to the host computer interface management information table 500 (FIG. 5) (step 606), a determination is made as to whether the host computer interface which is connected to the control processor is in the CKD or FBA format (step 607).

If it is in the CKD format, the control processor 305 converts the read data from FBA format into CKD format data (step 608) and transfers the converted data to the host computer 300 (step 609). Conversion of FBD data into CKD data will described later.

If it is in the FBA format, the data which is read out from the cache slot 312 is transferred to the host computer 303 without conducting any conversion.

Subsequently, the cache slot 312 is released (step 614) and DCB is released (step 615).

The write data on the cache slot 312 is written into the drives 315, 316 by the control processors 310, 311 asynchronously with the operation of the host computers 300, 303. In other words, the control processors 310, 311 retrieve the cache memory and when data to be written into the drive is found in the cache memory, it writes the data into the drive.

FIG. 7 is a flow chart showing the flow (700) of operation in DCB reservation operation (601) in FIG. 6.

Interprocessor exclusive information 402 of the DCB 400 corresponding to the specified drive number 401 is preset (step 701).

Subsequently, a determination is made as to whether the interhost exclusive information is "on" or not (step 702).

If it is not "on", the interhost exclusive information 403 is set "on" (step 703) to set success of reservation in a return code (step 704).

Then, the interprocessor exclusive information 402 is canceled (step 708) to end DCB reservation operation (step 709).

If the interhost exclusive information has already been "on", the operation will be conducted as follows:

The fact that the DCB is being used is reported to the host computer (step 705).

Subsequently, the host computer to which the use of DCB is reported is recorded in the DCB vacancy waiting information 404 of DCB 400 (step 706).

Failure of reservation is set in the return code (step 707) and then the interprocessor exclusive information 401 is canceled (step 708).

FIG. 8 is a flow chart showing the flow of DCB release processing (615) in FIG. 6.

Now, the interprocessor exclusive information 402 of DCB 400 of the drive number 401 in which release is requested is set (step 801).

Then, the interhost exclusive information 403 is canceled (step 802).

Then, a determination is made as to whether a host computer exists, which is registered in the vacancy waiting information of DCB (step 803).

If no host computer exists, the interprocessor exclusive information 402 is canceled (step 605) to end (step 806) the drive release operation (step 800).

If a host computer exists, vacancy of DCB is reported to the registered host computer (step 804).

This prevents the DCB from being predominantly used by either one of the host computers. Then, the interprocessor exclusive information is canceled (step 805).

FIG. 9 is a chart for explaining the conversion between CKD and FBA format data.

Referring now to FIG. 9, conversion from CKD format data to FBA format data will be briefly described.

Information on which block CKD of (1) and (2) are positioned is available. Information on data length of CKD is stored in a C (count) area. In the case of data input/output to/from a host having CKD format, the data length information is divided by 512 bytes to determine the number of necessary blocks. The data is front-packed from the front end of the positioned blocks. The remaining portion of the block is left as it is. In the case of data input/output to/from host computer having FBA format, the specified LBA (Logical Block Address) is accessed from the host computer.

Now, conversion from FBA format data to CKD format data will be described.

In the case of data input/output to/from a host computer having CKD format, a block having the C information of interest is searched and accessed from the specified C information.

In the case of data input/output to/from a host computer having FBA format a block of specified LBA is accessed.

Since the method of conversion is well known, its detailed description will be omitted herein.

Although a magnetic disk device is used as a storage medium in the above-mentioned embodiments, the above-mentioned data access processing can be implemented by using a magnetic tape device or floppy disk device in lieu of magnetic disk device.

The host computers may be added to or removed from the storage control or the computer system for each of control processors for processing a request of input/output of the host computer and control processors for processing input/output to/from the storage media. Further, host computers may be added to or removed from the storage control of a drive (storage) having a device input/output interface. Such an embodiment will be described with reference to FIG. 10. Since components like to those in FIG. 3 are designated by like reference numerals, description of them will be omitted.

In FIG. 10, a pair (319) of a fiber channel control 317 and a control processor 318 for controlling the same are connected (added) to or removed (deleted) from a common bus 308 of the disk control 302.

Further, a pair (322) of a floppy disk (FD) 321 and a control processor 320 for controlling the same are connected (added) to or removed from the common bus 308 of the disk control 302.

The service processor 317 updates the contents of the drive control block 318 in a common memory 307 and the host computer interface management table 319 in response to addition or deletion of the pairs 319 and 322.

In accordance with the present invention, data on storage media can be shared by host computers in which storage controls have various different host computer input/output interfaces as mentioned in the foregoing embodiments. Exensionability of file subsystem and the responsiveness of data is enhanced.

Since host computers having various different host computer input/output interfaces or various storage media for storing input/output data of the host computers can be connected by a single storage control, a wide range of configurations of computer system are possible.

What is claimed is:

1. A storage control for use in a computer system including a plurality of host computers having different kinds of computer input/output interfaces; a storage control for controlling data input/output to/from said host computers; and at least one storage medium having a device input/output interface for conducting data input/output in a given data

format for storing input/output data to/from said host computers,

said storage control comprising:

a plurality of control processors for controlling the transfer of input/output data to/from the host computers, each of said processors being connected to one of said plurality of host computers; and

a management table for managing the data format of the input/output interface of each of said host computer;

each of said control processors including a data format converting unit which, in response to a request of read or write of associated host computer, converts the format of the read or write data into said given data format when the data format from the associated host computer dose not match said given data format and does not convert the format of the read or write data when they match.

2. A storage control as defined in claim 1 in which the content in said management table is set/canceled in response to another processor to allow a desired number of host computers and associated control processors to be added or removed.

3. A storage control as defined in claim 2 and further including a drive control block for managing the state of access to said storage media by said host computers.

4. A storage control as defined in claim 3 in which the number of said storage media is arbitrary changed to allow a desired number of storage media is be added or removed.

5. A storage control as defined in claim 1 and further including a cache memory which is connected to said storage media so that input/output of data is conducted by said host computers via the cache memory.

6. A computer system comprising;

a plurality of host computers having different kinds of computer input/output interfaces; a storage control for controlling data input/output to/from said host computers; and

at least one storage medium having a device input/output interface for conducting data input/output in a given data format for storing input/output data to/from said host computers,

said storage control including:

a plurality of control processors for controlling the transfer of input/output data to/from the host computers, each of said processors being connected to one of said plurality of host computers; and

a management table for managing the data format of the input/output interface of each of said host computer;

each of said control processors including a data format converting unit which, in response to a request of read or write from associated host computer, converts the format of the read or write data into said given data format when the data format of the associated host computer does not match said given data format and does not to convert the format of the read or write data when they match.

7. A computer system as defined in claim 6 in which the content in said management table is set/canceled in response to another processor to allow a desired number of host computers and associated control processors to be added or removed.

8. A computer system as defined in claim 7 and further including a drive control block for managing the state of access to said storage media by said host computers.

9. A computer system as defined in claim 8 in which the number of said storage media is arbitrary changed to allow a desired number of storage media to be added or removed.

10. A computer system as defined in claim 6 and further including a cache memory which is connected to said storage media so that input/output of data is conducted by said host computers via the cache memory.

11. A storage control apparatus for controlling transfer of data between a plurality of host computers and at least one storage medium which stores data of a particular data format said particular data format being the same as a data format used by at least one of said host computers, said storage control apparatus comprising:

a plurality of control processors each controlling transfer of data between one of the host computers and the storage medium,

wherein each control processor includes a data format converting unit which, in response to a request to write data in the storage medium from a corresponding host computer, converts the data format of the write data into the particular data format when the data format of the write data does not match the particular data format and does not convert the data format of the write data into the particular data format when the data format of the write data matches the particular data format.

12. A storage control apparatus according to claim 11 wherein each data format converting unit of each control processor, in response to a request to read data in the storage medium from a corresponding host computer, converts the data format of the read data into a data format other than the particular data format used by the corresponding host computer when the data format used by the corresponding host computer does not match the particular data format and does not convert the data format of the read data into the particular data format when the data format used by the corresponding host computer matches the particular data format.

13. A storage control apparatus according to claim 11 further comprising:

a management table for managing information indicating the data format used by each of the host computers, wherein a data format of a host computer is determined by referring to said information.

14. A storage control apparatus according to claim 13, said information stored in said management table is set/cancelled in response to addition or removal of a host computer and a corresponding control processor.

15. A storage control apparatus according to claim 14, further comprising:

a drive control block for managing the state of access to said storage media by said host computers.

16. A storage control apparatus according to claim 15, wherein the number of said storage media is arbitrarily changed to add or remove storage media as desired.

17. A storage control according to claim 11, further comprising:

a cache memory which is connected to said storage media so that input/output of data is conducted by said host computers via the cache memory.

18. A method of controlling transfer of data between a plurality of host computers and at least one storage medium which stores data of a particular data format, said particular data format being the same as a data format used by at least one of said host computers, said method comprising the steps of:

controlling transfer of data between one of the host computers and the storage medium; and

in response to a request to write data in the storage medium from a corresponding host computer, converting the data format of the write data into the particular data format when the data format of the write data does not match the particular data format and not converting the data format of the write data into the particular data format when the data format of the write data matches the particular data format.

19. A method according to claim 18 further comprising the step of:

in response to a request to read data in the storage medium from a corresponding host computer, converting the data format of the read data into a data format other than the particular data format used by the corresponding host computer when the data format used by the host computer does not match the particular data format and not converting the data format of the read data into the particular data format when the data format used by the host computer matches the particular data format.

* * * * *

US006651130B1

(12) **United States Patent**
Thibault

(10) **Patent No.:** **US 6,651,130 B1**
(45) **Date of Patent:** **Nov. 18, 2003**

(54) **DATA STORAGE SYSTEM HAVING SEPARATE DATA TRANSFER SECTION AND MESSAGE NETWORK WITH BUS ARBITRATION**

(75) Inventor: **Robert Thibault**, Westboro, MA (US)

(73) Assignee: **EMC Corporation**, Hopkinton, MA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/561,532**

(22) Filed: **Apr. 28, 2000**

(51) **Int. Cl.**^{7} **G06F 1/00**

(52) **U.S. Cl.** **710/317**; 710/316; 710/309; 710/29; 710/31; 711/112; 711/114

(58) **Field of Search** 710/317, 316, 710/309, 306, 311, 305, 29, 31, 38, 60, 243; 711/112, 111, 113, 114, 117, 119, 131, 148, 149, 168

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,480,307 A		10/1984	Budde et al.	364/200
4,484,262 A		11/1984	Sullivan et al.	364/200
4,636,948 A		1/1987	Gdaniec et al.	
4,800,483 A		1/1989	Yamamoto et al.	364/200
5,119,485 A		6/1992	Ledbetter, Jr. et al.	395/425
5,166,674 A		11/1992	Baum et al.	
5,206,939 A		4/1993	Yanai et al.	
5,214,768 A		5/1993	Martin et al.	395/425
5,269,011 A		12/1993	Yanai et al.	395/425
5,335,327 A		8/1994	Hisano et al.	395/250
5,479,611 A		12/1995	Oyama	395/185.01
5,689,728 A		11/1997	Sugimoto et al.	395/858
5,742,789 A	*	4/1998	Ofer et al.	711/118
5,799,209 A		8/1998	Chatter	
5,805,821 A		9/1998	Saxena et al.	395/200.61
5,813,024 A		9/1998	Saito	
5,819,054 A		10/1998	Ninomiya et al.	395/308
5,819,104 A	*	10/1998	Tuccio	710/72
5,890,207 A		3/1999	Sne et al.	711/113
6,009,481 A		12/1999	Mayer	710/33
6,038,638 A		3/2000	Cadden et al.	711/111
6,061,274 A	*	5/2000	Thibault et al.	365/189.05
6,081,860 A		6/2000	Bridges et al.	
6,125,429 A	*	9/2000	Goodwin et al.	711/143
6,134,624 A		10/2000	Burns et al.	710/131

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

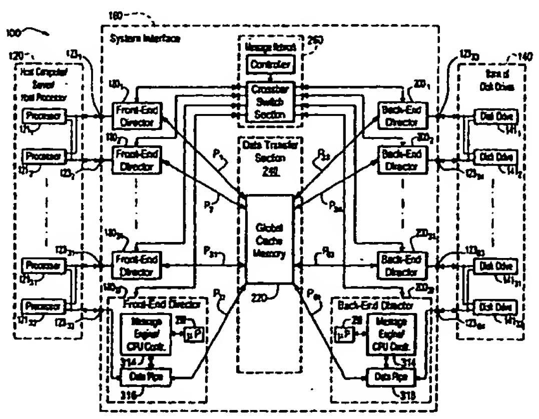
GB	2 325 541 A	11/1998
GB	2 360 377 A	9/2001
GB	2366425	10/2001
WO	93/01553	1/1993
WO	WO 99/26150	5/1999

Primary Examiner—Paul R. Meyers
Assistant Examiner—Raymond N Phan
(74) *Attorney, Agent, or Firm*—Daly, Crowley & Mofford, LLP

(57) **ABSTRACT**

A system interface includes a plurality of first directors, a plurality of second directors, a data transfer section and a message network. The data transfer section includes a cache memory. The cache memory is coupled to the plurality of first and second directors. The messaging network operates independently of the data transfer section and such network is coupled to the plurality of first directors and the plurality of second directors. The first and second directors control data transfer between the first directors and the second directors in response to messages passing between the first directors and the second directors through the messaging network to facilitate data transfer between first directors and the second directors. The data passes through the cache memory in the data transfer section. A method for operating a data storage system adapted to transfer data between a host computer/server and a bank of disk drives. The method includes transferring messages through a messaging network with the data being transferred between the host computer/server and the bank of disk drives through a cache memory, such message network being independent of the cache memory.

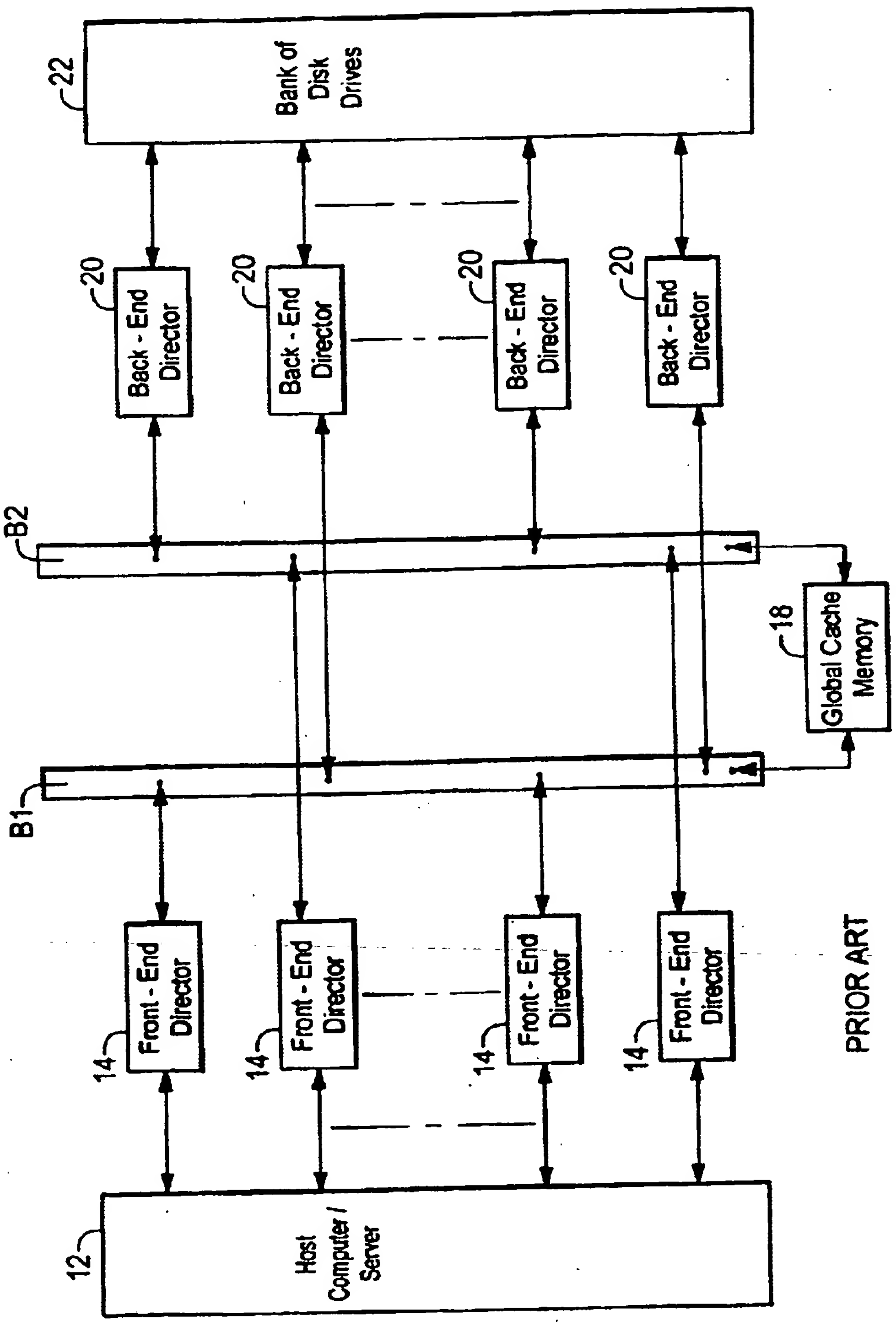
12 Claims, 25 Drawing Sheets



U.S. PATENT DOCUMENTS

6,178,466 B1		1/2001	Gilbertson et al.	....	170/3
6,230,229 B1	*	5/2001	Van Krevelen et al.	.....	710/317
6,275,877 B1		8/2001	Duda	....	710/236
6,317,805 B1	*	11/2001	Chilton et al.	....	710/311
6,378,029 B1	*	4/2002	Venkitakrishnan et al.	.	710/317
6,389,494 B1	*	5/2002	Walton et al.	....	710/305
6,397,281 B1		5/2002	MacLellan et al.	....	710/113

* cited by examiner



PRIOR ART

FIG. 1

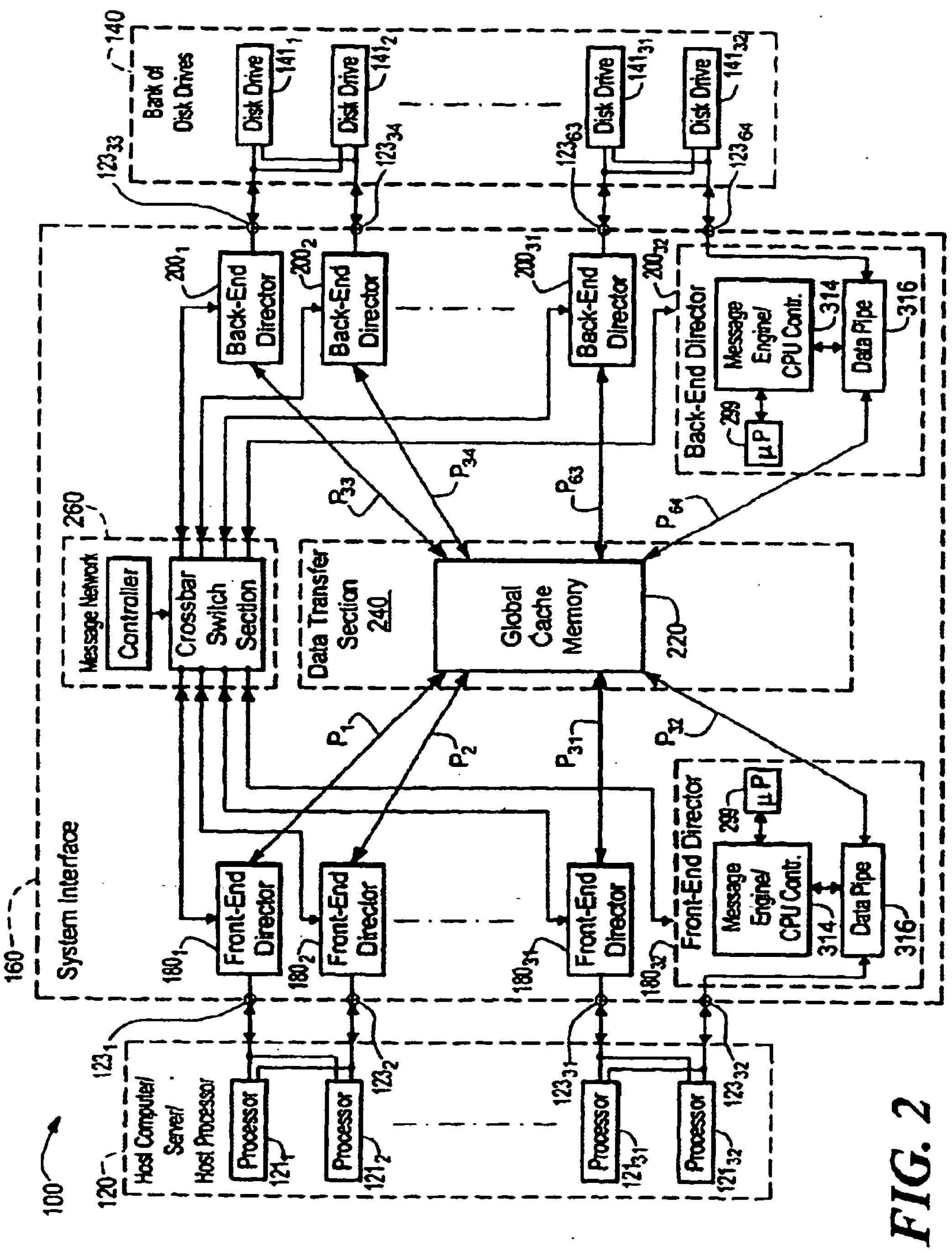

100
120
Host Computer/ Server/ Host Processor
Processor
121_1
121_2
121_{31}
121_{32}
123_1
123_2
123_{31}
123_{32}
160
System Interface
Front-End Director
180_1
180_2
180_{31}
180_{32}
Message Network
Controller
Crossbar Switch Section
260
Data Transfer Section 240
Global Cache Memory
220
P_1
P_2
P_{31}
P_{32}
P_{33}
P_{34}
P_{63}
P_{64}
Message Engine/ CPU Contr.
Data Pipe
314
316
299
μP
Back-End Director
200_1
200_2
200_{31}
200_{32}
123_{33}
123_{34}
123_{63}
123_{64}
Bank of Disk Drives
140
Disk Drive
141_1
141_2
141_{31}
141_{32}

FIG. 2

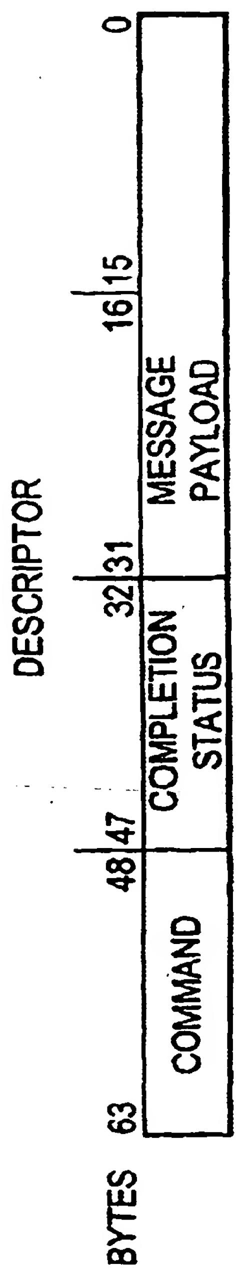

DESCRIPTOR
BYTES
63
48
47
32
31
16
15
0
COMMAND
COMPLETION STATUS
MESSAGE PAYLOAD

FIG. 2A

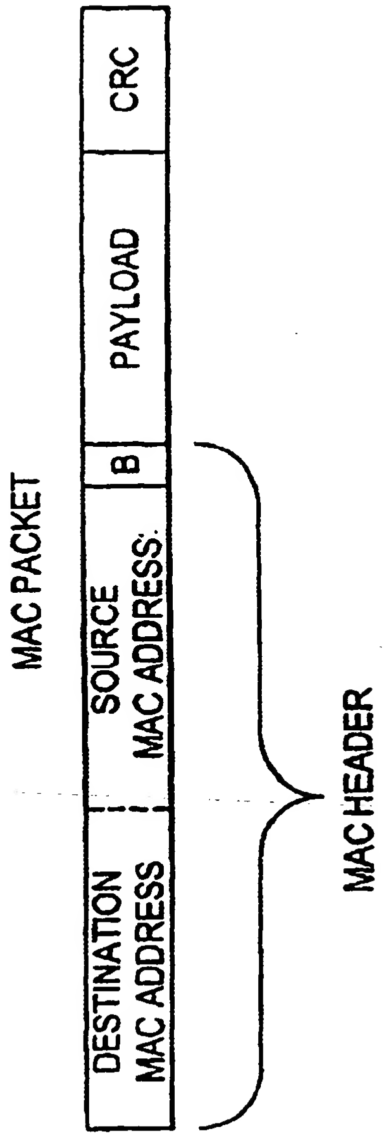

MAC PACKET
DESTINATION MAC ADDRESS
SOURCE MAC ADDRESS
B
PAYLOAD
CRC
MAC HEADER

FIG. 2B

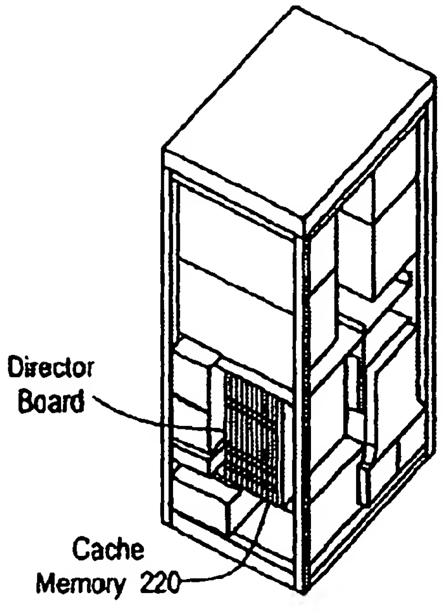

Director Board
Cache Memory 220

FIG. 3

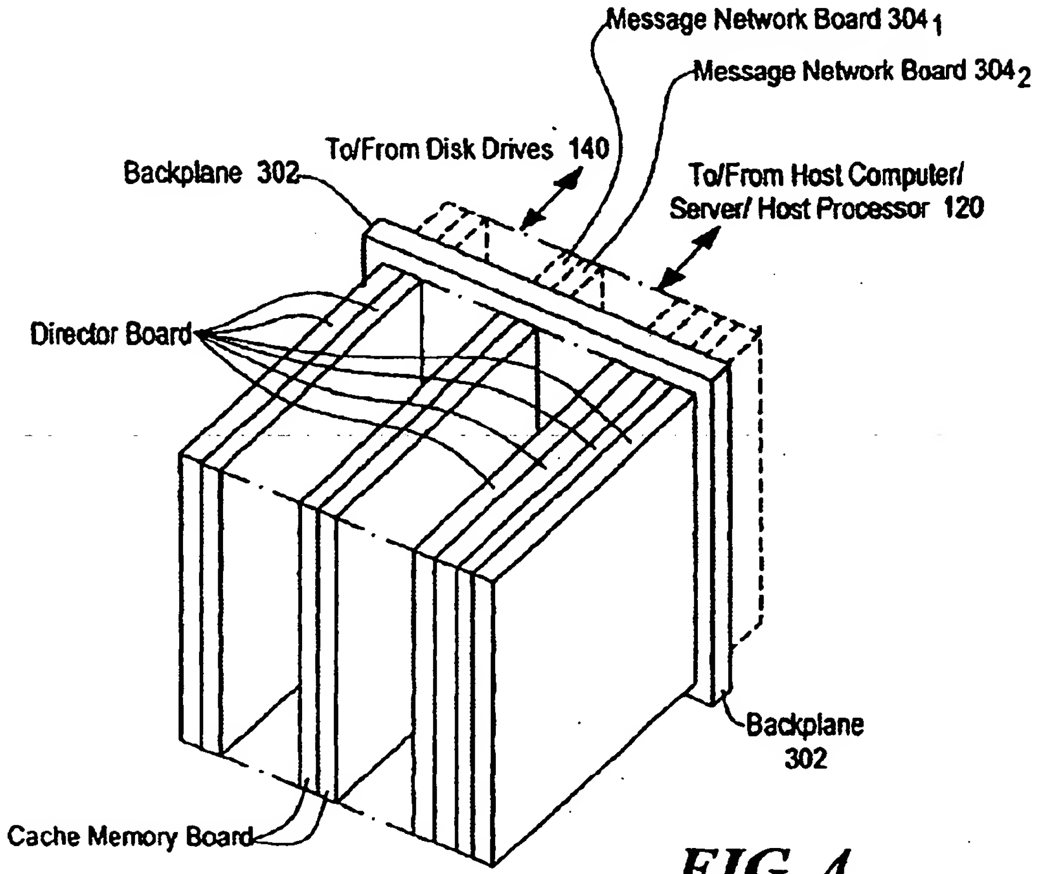
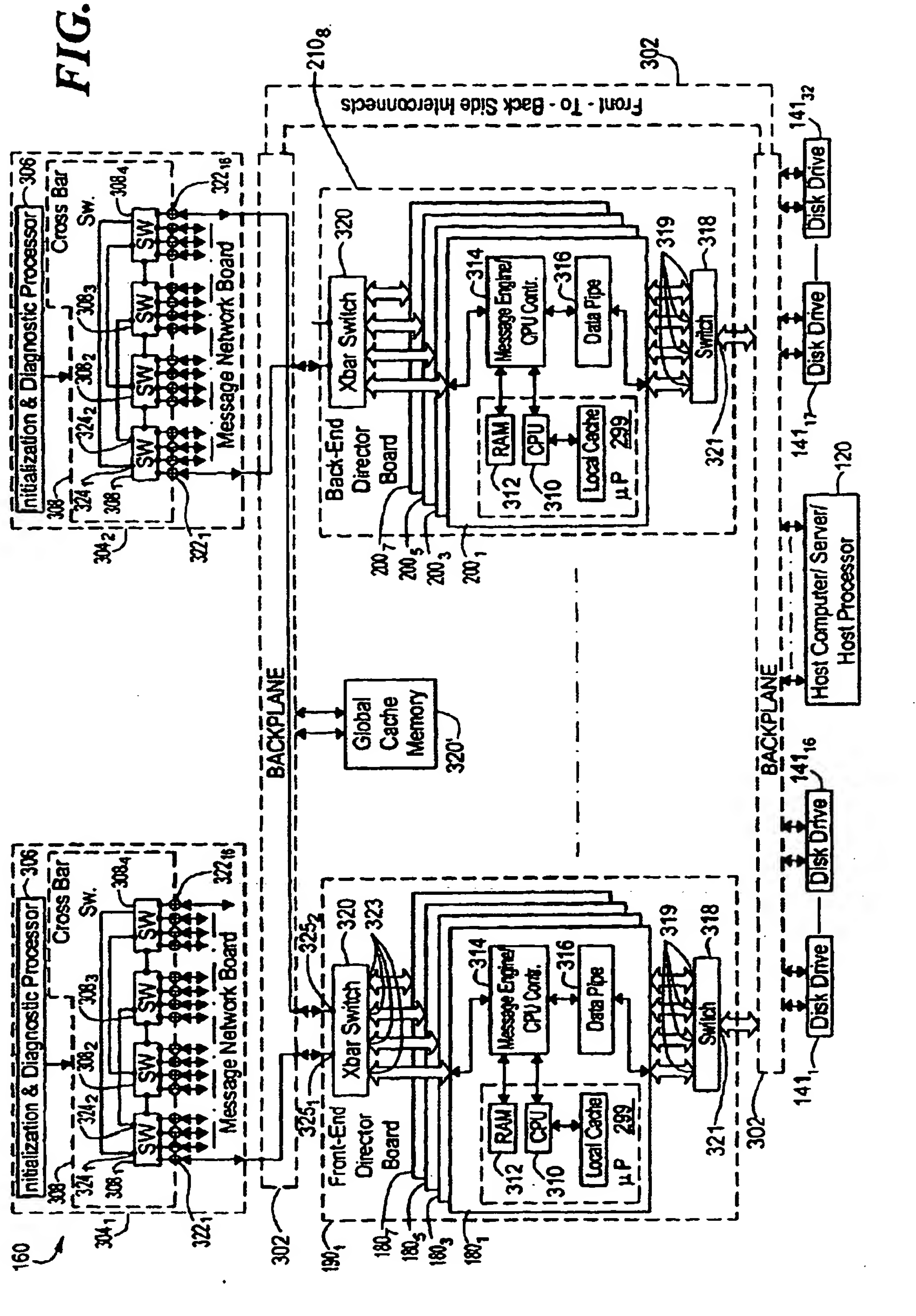

Message Network Board 304_1
Message Network Board 304_2
To/From Disk Drives 140
Backplane 302
To/From Host Computer/ Server/ Host Processor 120
Director Board
Backplane 302
Cache Memory Board

FIG. 4

FIG. 5


160
Initialization & Diagnostic Processor
306
Cross Bar Sw.
SW
308_1
308_2
308_3
308_4
309
324
324_2
304_1
304_2
322_1
322_{16}
Message Network Board
BACKPLANE
302
325_1
325_2
Xbar Switch
320
323
Front-End Director Board
Back-End Director Board
190_1
210_8
180_1
180_3
180_5
180_7
200_1
200_3
200_5
200_7
Message Engine/ CPU Contr.
314
Data Pipe
316
RAM
CPU
Local Cache
µP
299
312
310
Switch
318
319
321
Global Cache Memory
320'
Front - To - Back Side Interconnects
Disk Drive
141_1
141_{16}
141_{17}
141_{32}
Host Computer/ Server/ Host Processor
120

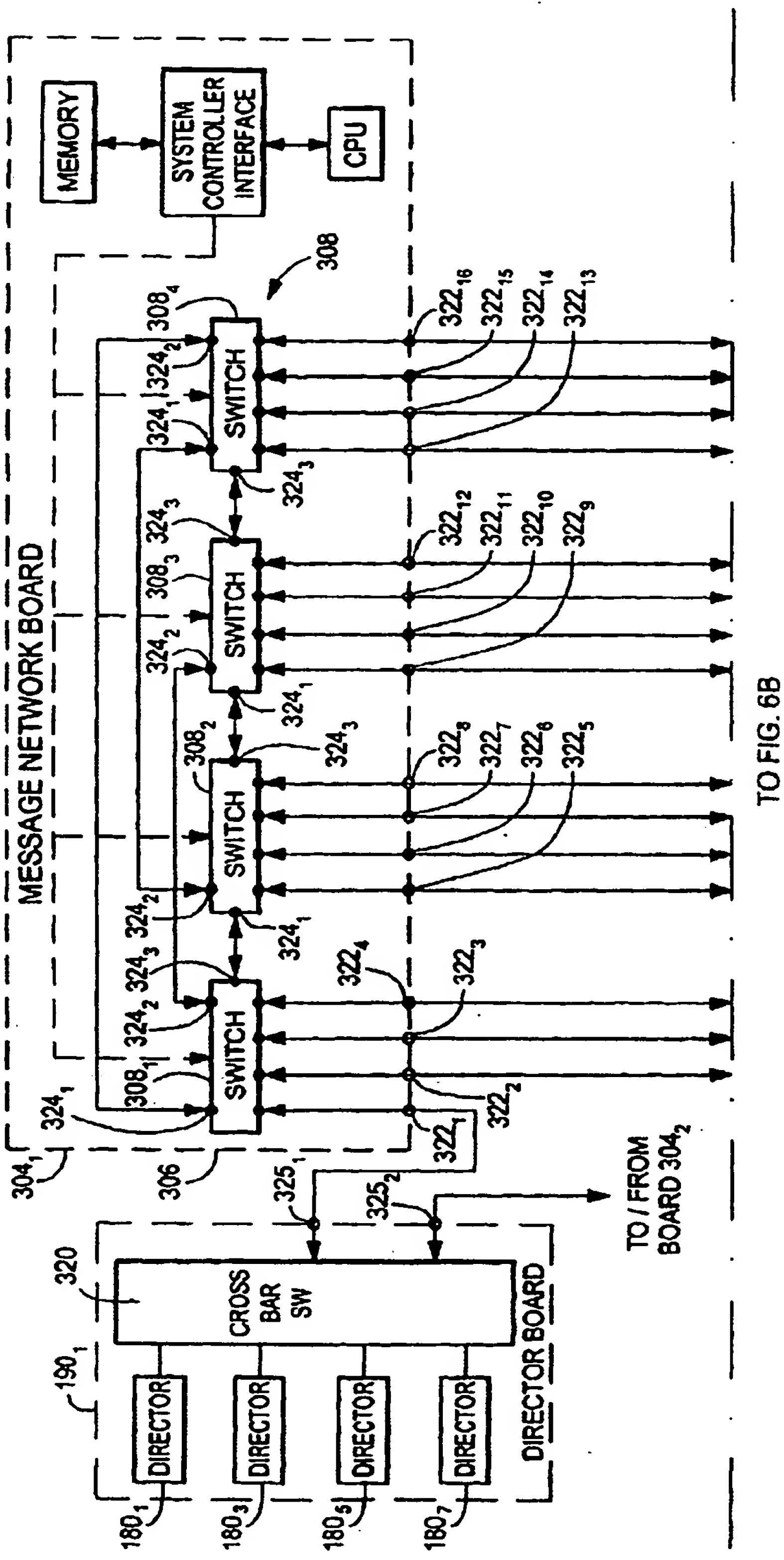

MESSAGE NETWORK BOARD
MEMORY
SYSTEM CONTROLLER INTERFACE
CPU
SWITCH
SWITCH
SWITCH
SWITCH
308
308_1
308_2
308_3
308_4
324_1
324_2
324_3
322_1
322_2
322_3
322_4
322_5
322_6
322_7
322_8
322_9
322_{10}
322_{11}
322_{12}
322_{13}
322_{14}
322_{15}
322_{16}
304_1
306
325_1
325_2
TO/FROM BOARD 304_2
TO FIG. 6B
320
CROSS BAR SW
190_1
DIRECTOR
DIRECTOR
DIRECTOR
DIRECTOR
DIRECTOR BOARD
180_1
180_3
180_5
180_7

FIG. 6A

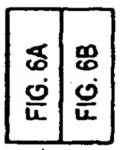


FIG. 6A
FIG. 6B

FIG. 6

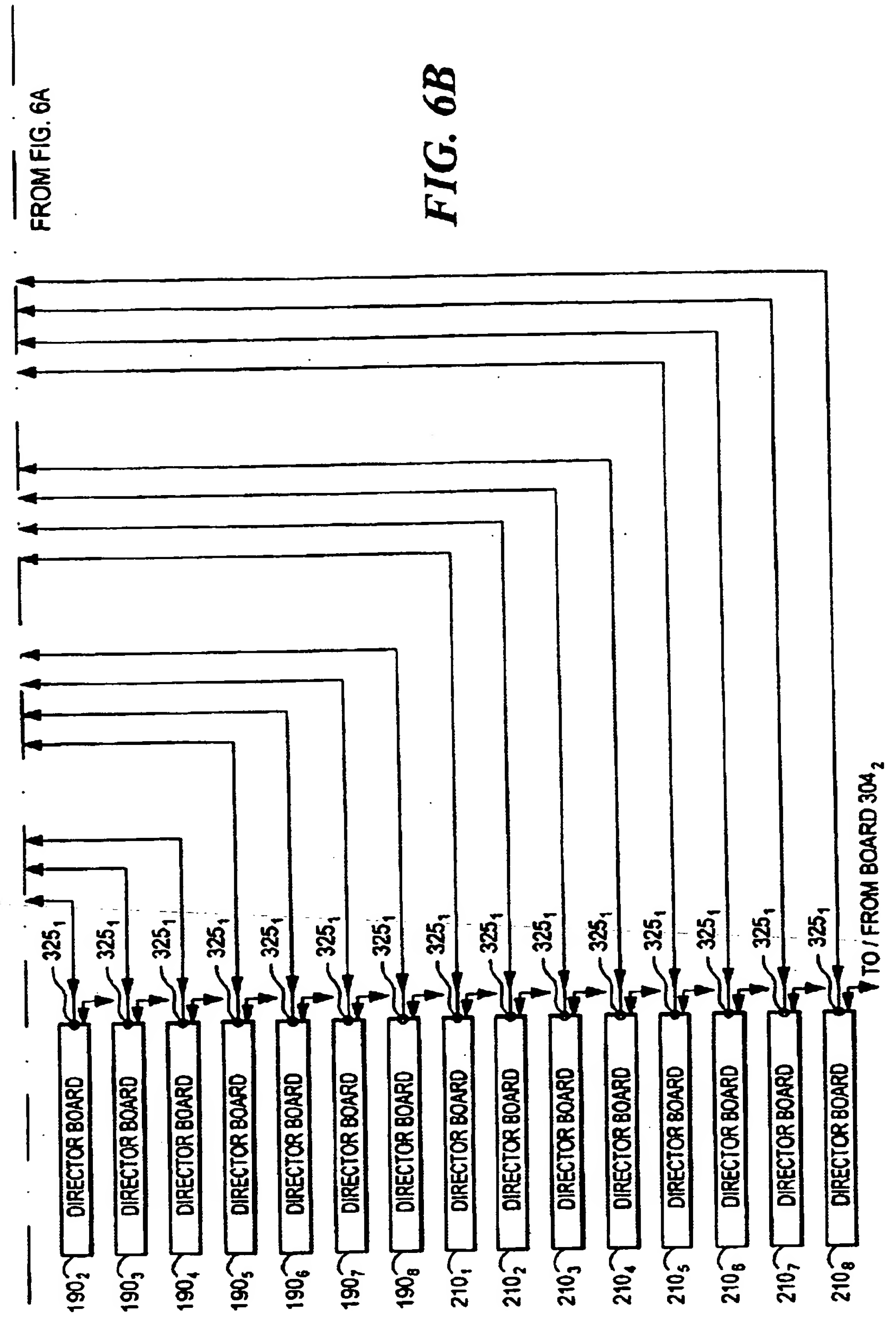

FROM FIG. 6A
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
DIRECTOR BOARD
190_2
190_3
190_4
190_5
190_6
190_7
190_8
210_1
210_2
210_3
210_4
210_5
210_6
210_7
210_8
325_1
TO / FROM BOARD 304_2

FIG. 6B

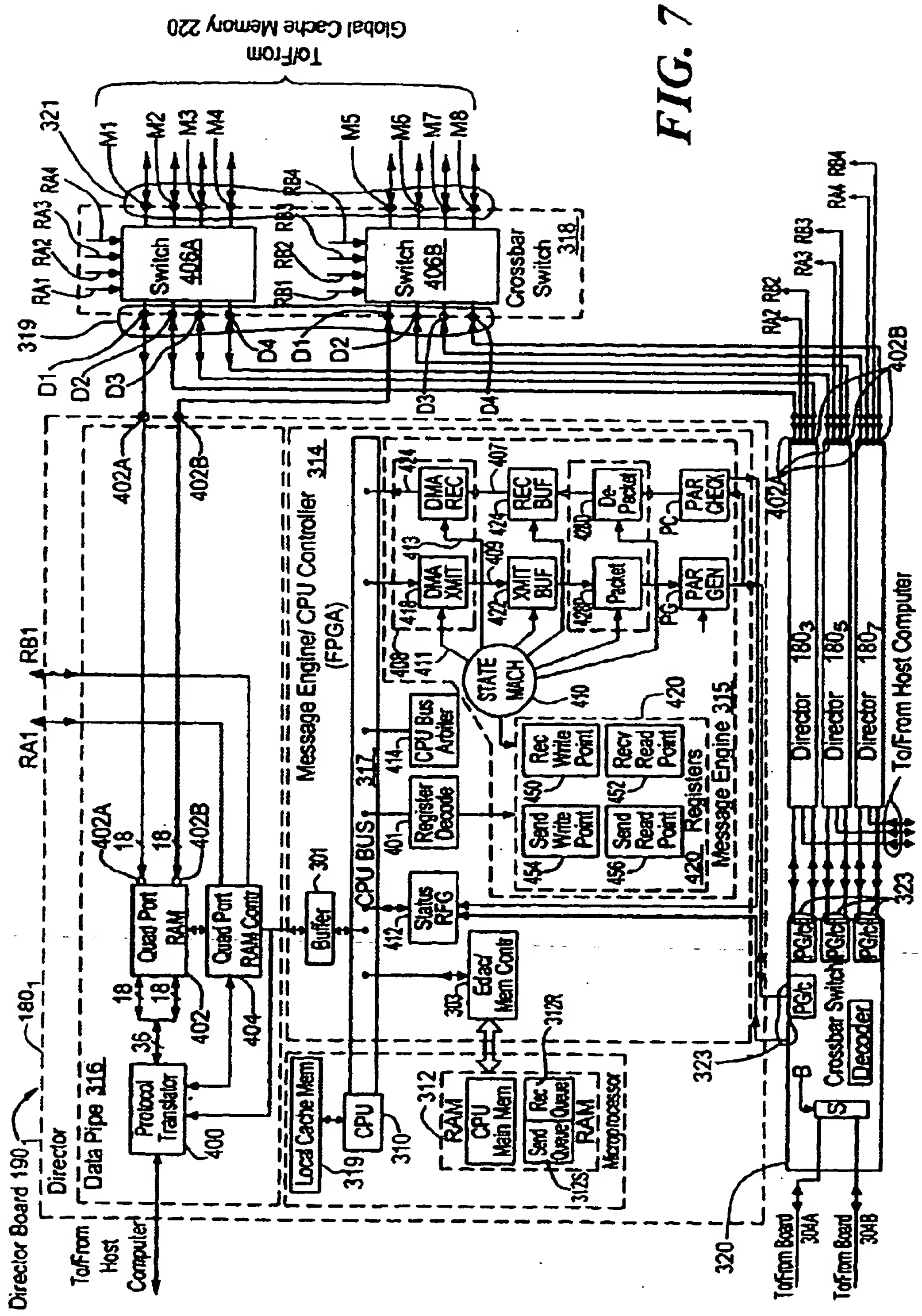

To/From
Global Cache Memory 220
M1
M2
M3
M4
M5
M6
M7
M8
321
RA1 RA2 RA3 RA4
Switch 406A
RB1 RB2 RB3 RB4
Switch 406B
Crossbar Switch 318
319
D1
D2
D3
D4
RA1
RB1
402A
402B
18
Quad Port RAM
Quad Port RAM Contr
Protocol Translator
Data Pipe 316
Director
Director Board 190
180₁
To/From Host Computer
36
400
402
404
Message Engine/ CPU Controller 314 (FPGA)
Buffer
301
CPU BUS 317
Status REG
412
Register Decode
401
CPU Bus Arbiter
414
DMA XMIT
DMA REC
XMIT BUF
REC BUF
Packet
De-Packet
PAR GEN
PAR CHECK
PC
STATE MACH
408
411
413
416
418
407
409
410
422
424
428
430
Send Write Point
Rec Write Point
Send Read Point
Recv Read Point
450
452
454
456
420 Registers
Message Engine 315
Edac/ Mem Contr
303
312R
CPU Main Mem
Send Queue
Rec Queue
RAM
312
312S
Microprocessor
CPU
310
Local Cache Mem
319
323
320
Director 180₃
Director 180₅
Director 180₇
To/From Host Computer
PGIc
Crossbar Switch
Decoder
B
S
To/From Board 304A
To/From Board 304B

FIG. 7

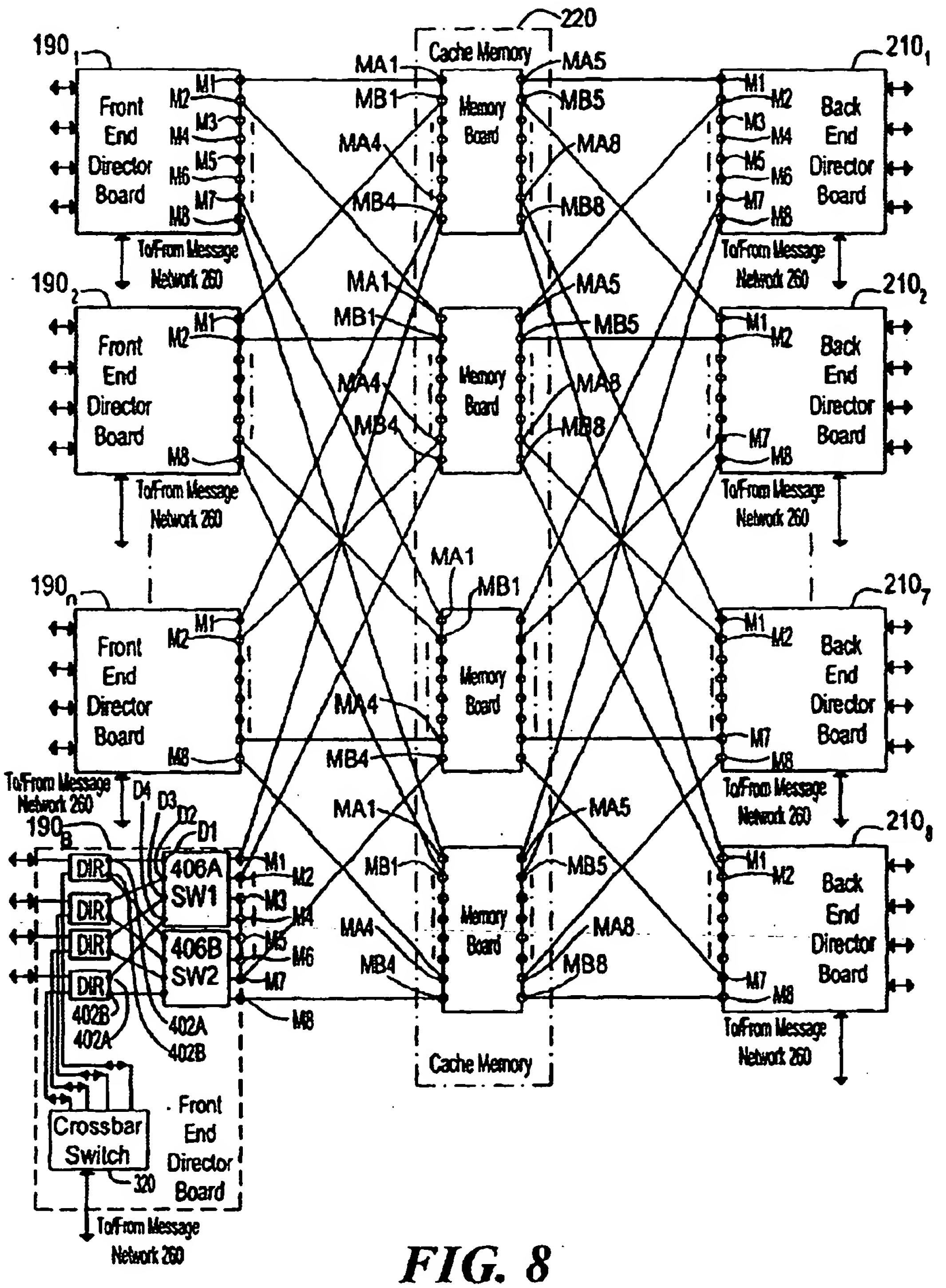

220
Cache Memory
Memory Board
MA1
MB1
MA4
MB4
MA5
MB5
MA8
MB8
190_1
190_2
190_n
190_8
Front End Director Board
M1
M2
M3
M4
M5
M6
M7
M8
To/From Message Network 260
210_1
210_2
210_7
210_8
Back End Director Board
D4
D3
D2
D1
DIR
406A SW1
406B SW2
402B
402A
Crossbar Switch
320

FIG. 8

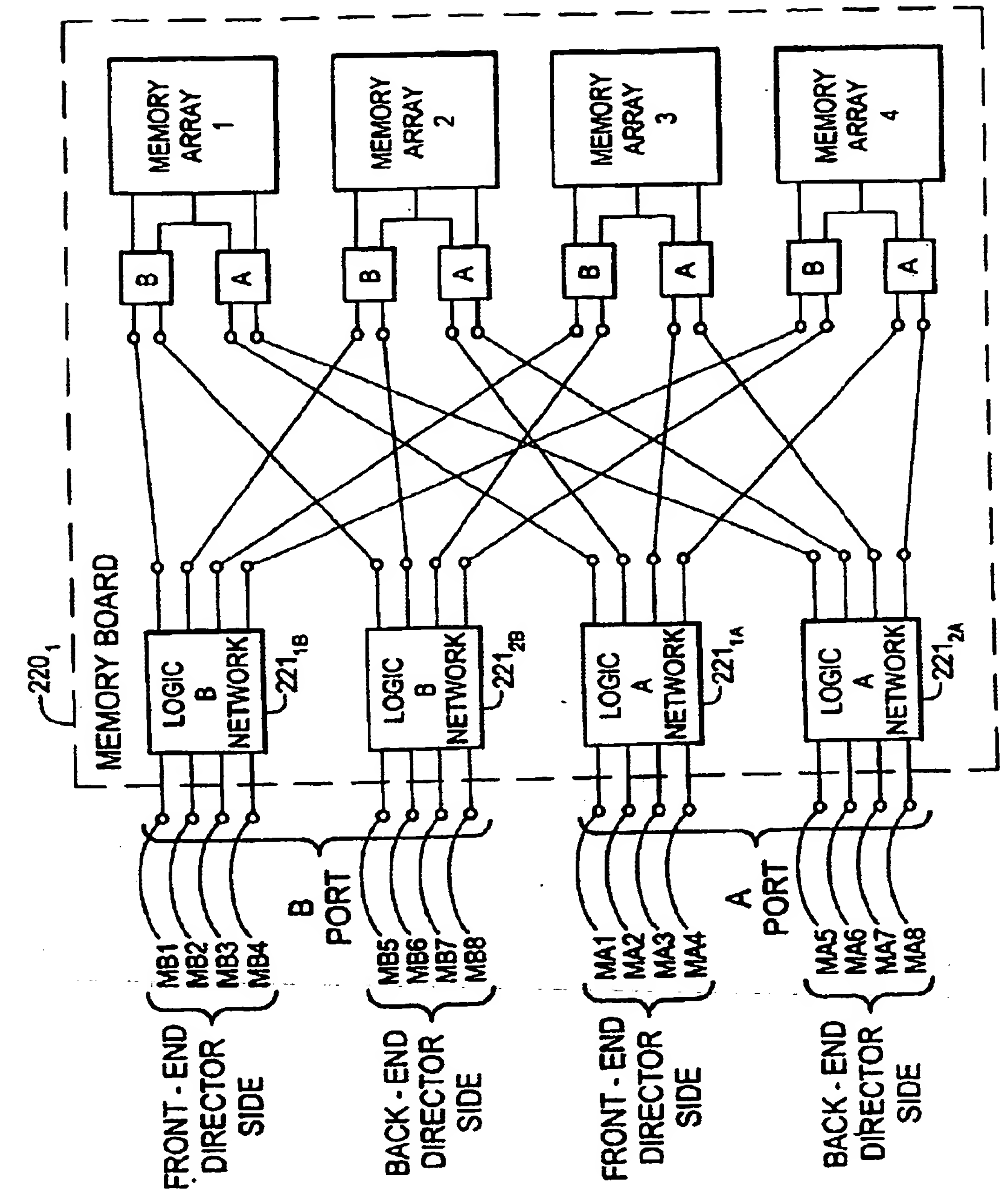

MEMORY ARRAY 1
MEMORY ARRAY 2
MEMORY ARRAY 3
MEMORY ARRAY 4
B
A
B
A
B
A
B
A
220₁
MEMORY BOARD
LOGIC B NETWORK
221₁B
LOGIC B NETWORK
221₂B
LOGIC A NETWORK
221₁A
LOGIC A NETWORK
221₂A
MB1
MB2
MB3
MB4
B PORT
MB5
MB6
MB7
MB8
MA1
MA2
MA3
MA4
A PORT
MA5
MA6
MA7
MA8
FRONT-END DIRECTOR SIDE
BACK-END DIRECTOR SIDE
FRONT-END DIRECTOR SIDE
BACK-END DIRECTOR SIDE

FIG. 8A

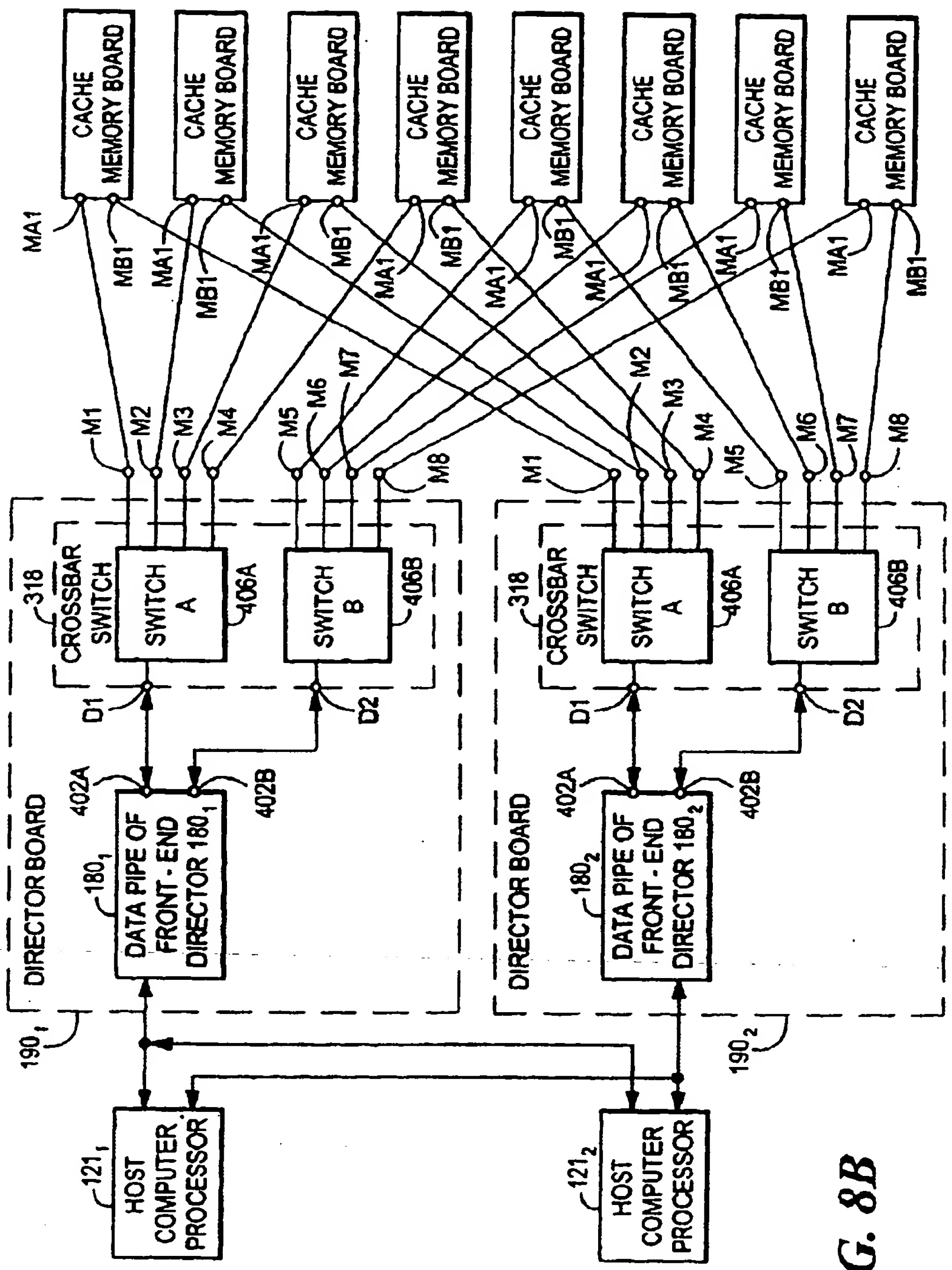

CACHE MEMORY BOARD
MA1
MB1
M1
M2
M3
M4
M5
M6
M7
M8
318
CROSSBAR SWITCH
SWITCH A
406A
SWITCH B
406B
D1
D2
402A
402B
DIRECTOR BOARD
180_1
DATA PIPE OF FRONT - END DIRECTOR 180_1
190_1
121_1
HOST COMPUTER PROCESSOR
180_2
DATA PIPE OF FRONT - END DIRECTOR 180_2
190_2
121_2

FIG. 8B

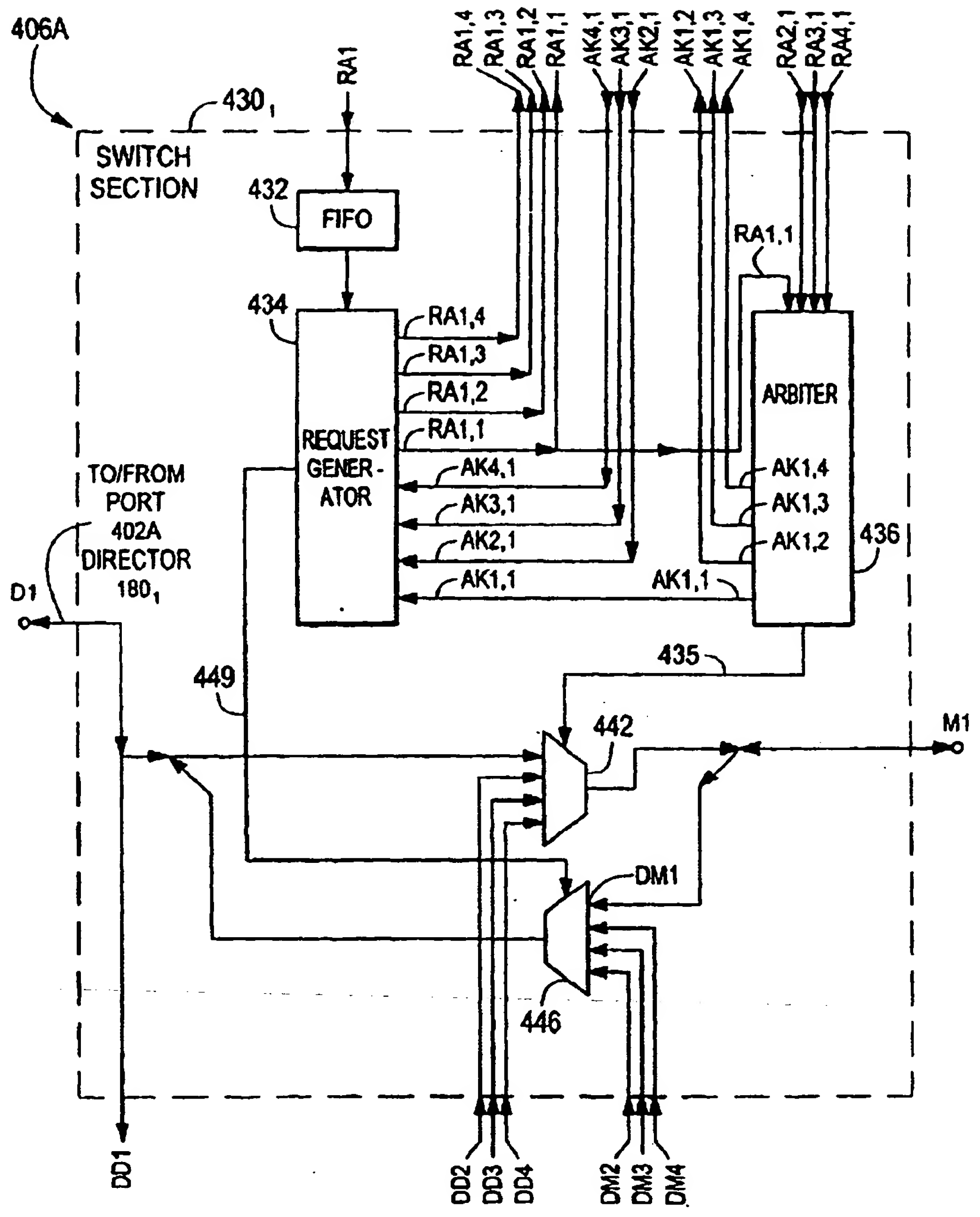


FIG. 8C-1

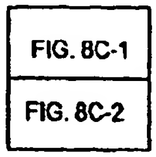


FIG. 8C

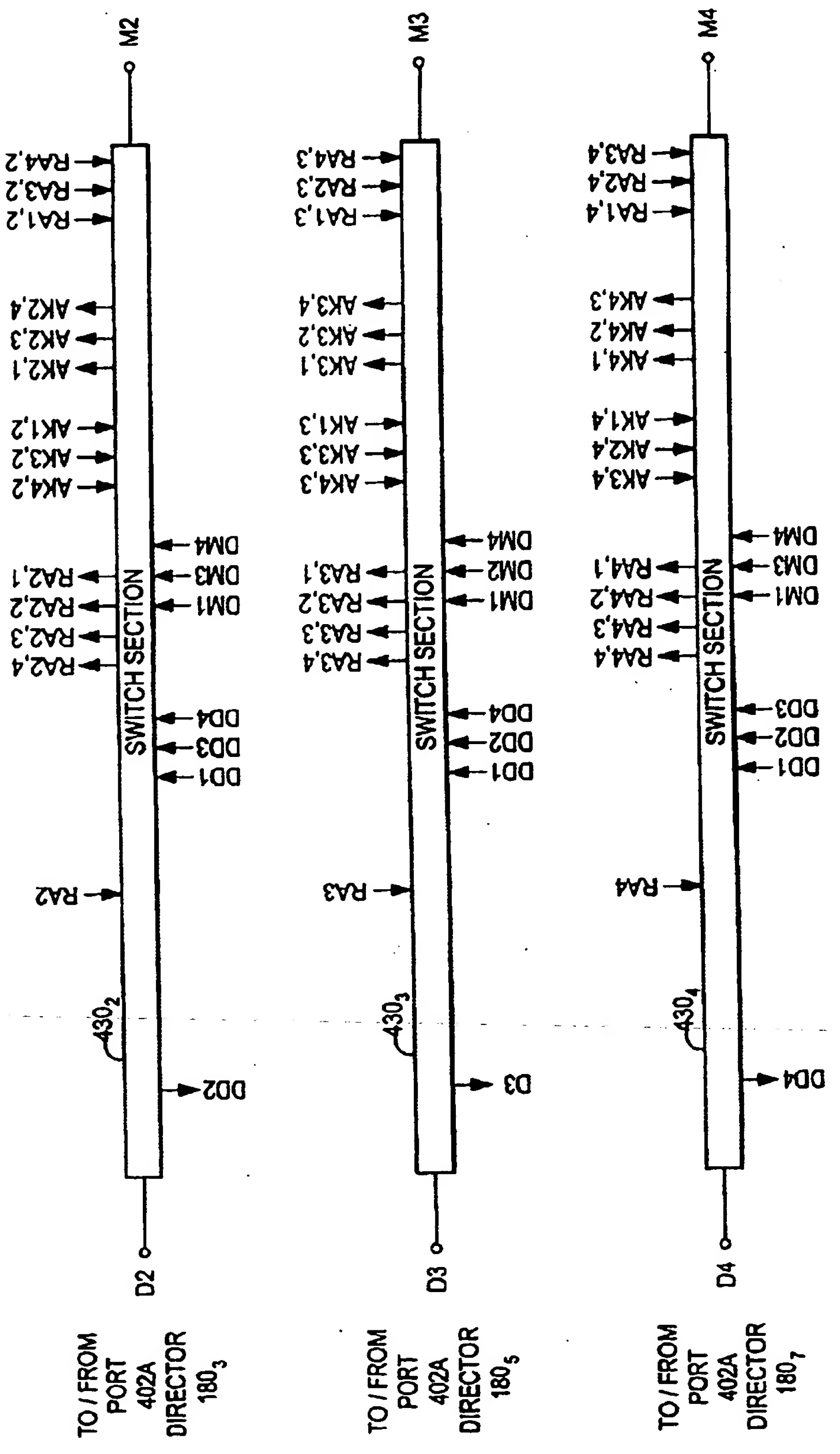
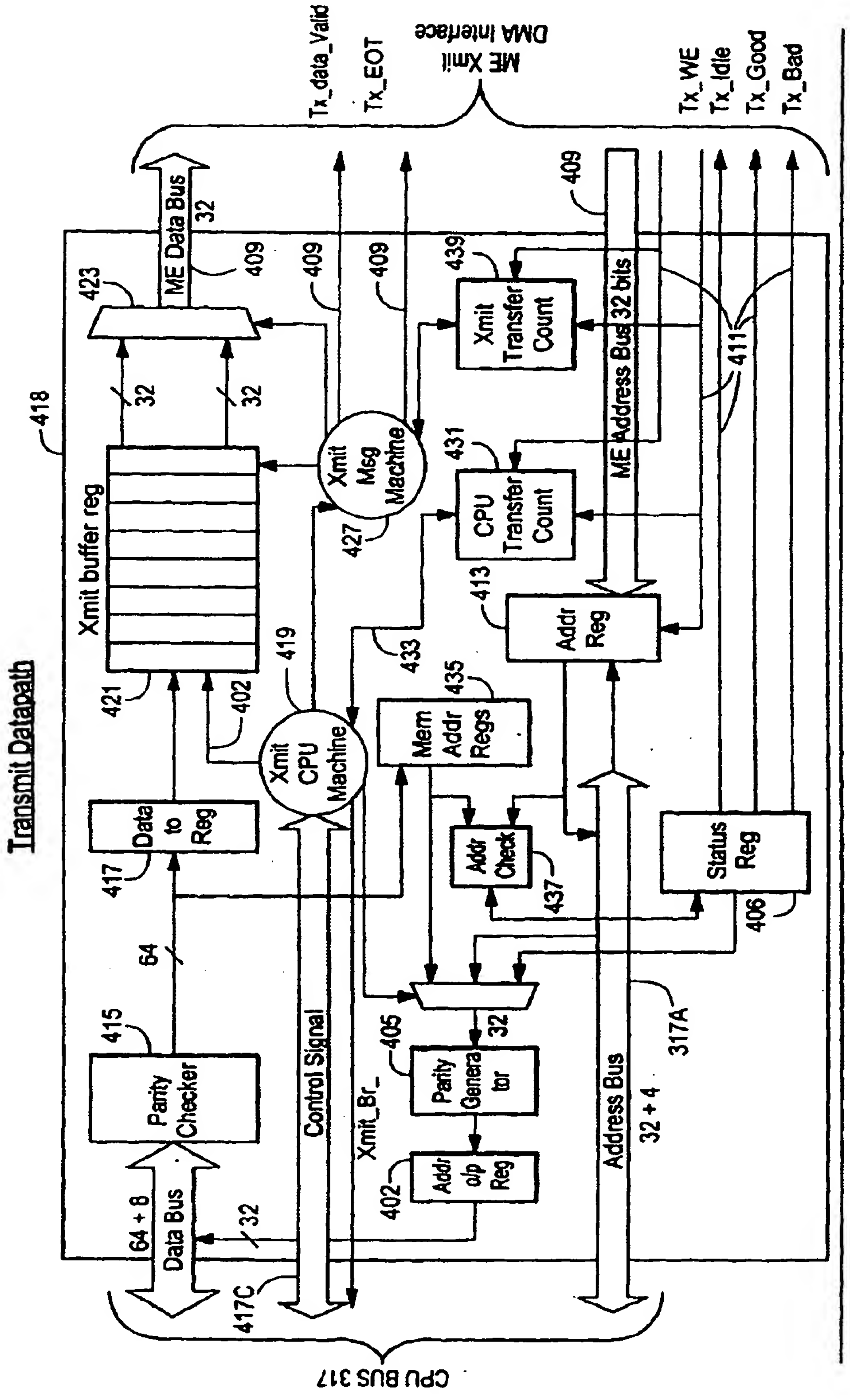

M2
M3
M4
RA4,2
RA3,2
RA1,2
AK2,4
AK2,3
AK2,1
AK1,2
AK3,2
AK4,2
DM4
DM3
DM1
RA2,1
RA2,2
RA2,3
RA2,4
SWITCH SECTION
DD4
DD3
DD1
RA2
430_2
DD2
D2
TO / FROM PORT 402A DIRECTOR 180_3
RA4,3
RA2,3
RA1,3
AK3,4
AK3,2
AK3,1
AK1,3
AK3,3
AK4,3
DM4
DM2
DM1
RA3,1
RA3,2
RA3,3
RA3,4
SWITCH SECTION
DD4
DD2
DD1
RA3
430_3
D3
D3
TO / FROM PORT 402A DIRECTOR 180_5
RA3,4
RA2,4
RA1,4
AK4,3
AK4,2
AK4,1
AK1,4
AK2,4
AK3,4
DM4
DM3
DM1
RA4,1
RA4,2
RA4,3
RA4,4
SWITCH SECTION
DD3
DD2
DD1
RA4
430_4
DD4
D4
TO / FROM PORT 402A DIRECTOR 180_7

FIG. 8C-2



Transmit Datapath
418
Xmit buffer reg
421
402
Data to Reg
417
Parity Checker
415
64
64 + 8
Data Bus
32
417C
CPU BUS 317
Control Signal
Xmit CPU Machine
419
Xmit Msg Machine
427
423
ME Data Bus
32
409
Tx_data_Valid
Tx_EOT
ME Xmit DMA Interface
Xmit_Br_
Mem Addr Regs
435
433
Addr Check
437
Parity Generator
405
Addr o/p Reg
402
CPU Transfer Count
431
Xmit Transfer Count
439
Addr Reg
413
ME Address Bus 32 bits
Address Bus
32 + 4
317A
Status Reg
406
411
Tx_WE
Tx_Idle
Tx_Good
Tx_Bad

FIG. 9

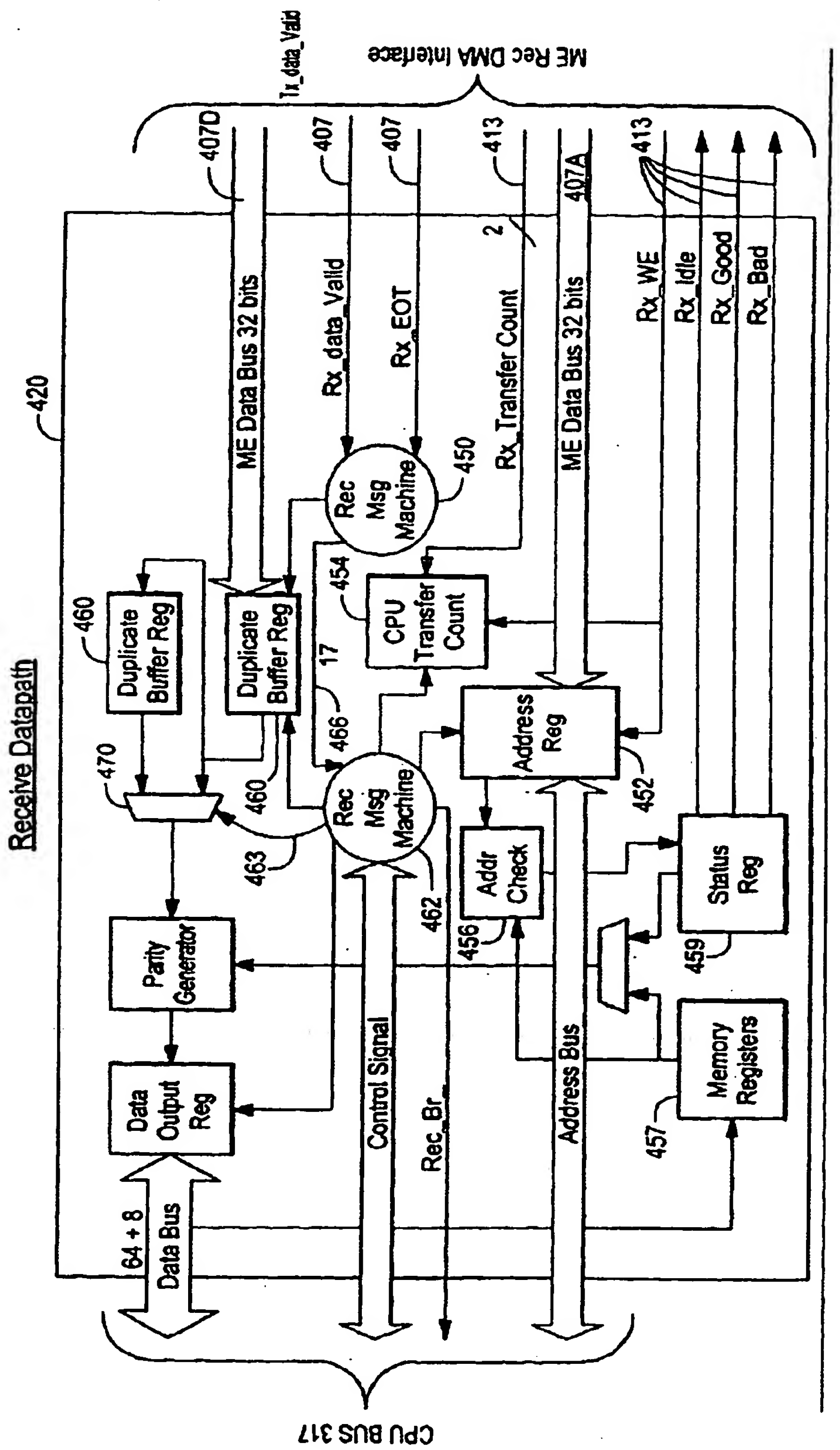
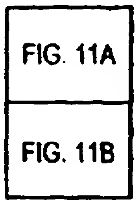
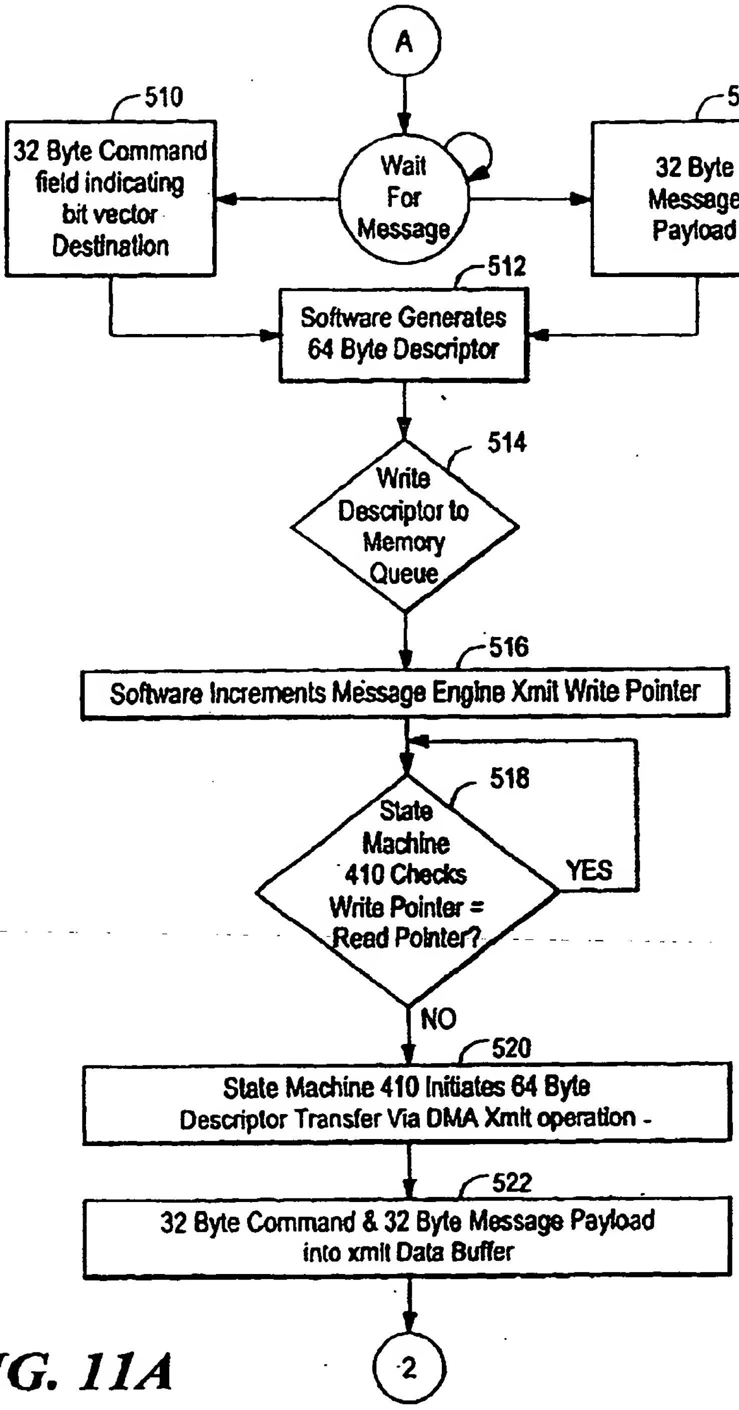

Receive Datapath
420
ME Rec DMA Interface
Tx_data_Valid
407D
407
407
413
407A
413
ME Data Bus 32 bits
Rx_data_Valid
Rx_EOT
Rx_Transfer Count
2
ME Data Bus 32 bits
Rx_WE
Rx_Idle
Rx_Good
Rx_Bad
Rec Msg Machine
450
Duplicate Buffer Reg
460
Duplicate Buffer Reg
454
CPU Transfer Count
17
466
460
470
Rec Msg Machine
462
463
Address Reg
452
Addr Check
456
Status Reg
459
Memory Registers
457
Parity Generator
Data Output Reg
64 + 8
Data Bus
Control Signal
Rec_Br_
Address Bus
CPU BUS 317

FIG. 10

FIG. 11





A
510
32 Byte Command field indicating bit vector Destination
Wait For Message
500
32 Byte Message Payload
512
Software Generates 64 Byte Descriptor
514
Write Descriptor to Memory Queue
516
Software Increments Message Engine Xmit Write Pointer
518
State Machine 410 Checks Write Pointer = Read Pointer?
YES
NO
520
State Machine 410 Initiates 64 Byte Descriptor Transfer Via DMA Xmit operation
522
32 Byte Command & 32 Byte Message Payload into xmit Data Buffer
2

FIG. 11A

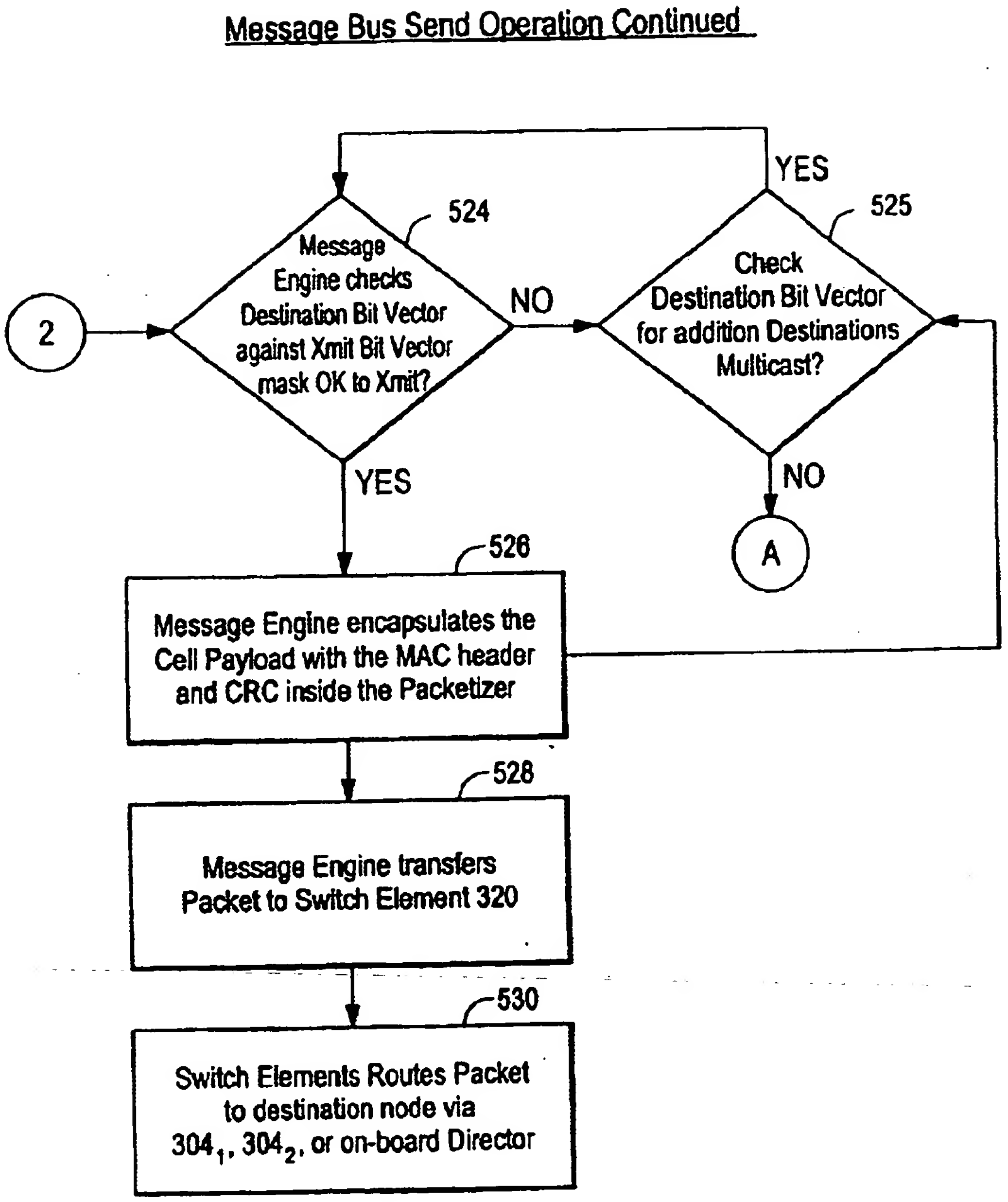

Message Bus Send Operation Continued
2
524
Message Engine checks Destination Bit Vector against Xmit Bit Vector mask OK to Xmit?
NO
YES
525
Check Destination Bit Vector for addition Destinations Multicast?
YES
NO
A
526
Message Engine encapsulates the Cell Payload with the MAC header and CRC inside the Packetizer
528
Message Engine transfers Packet to Switch Element 320
530
Switch Elements Routes Packet to destination node via 304₁, 304₂, or on-board Director

FIG. 11B

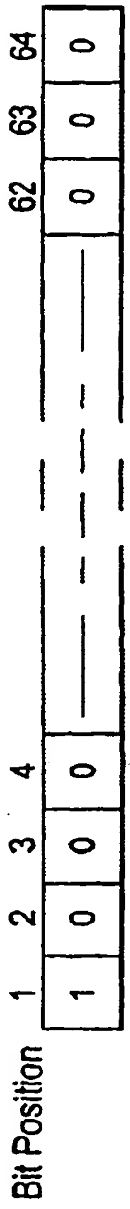


FIG. 11C

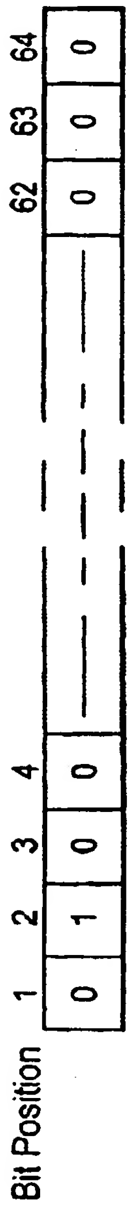


FIG. 11D

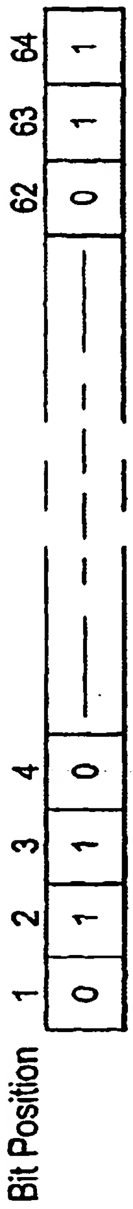


FIG. 11E

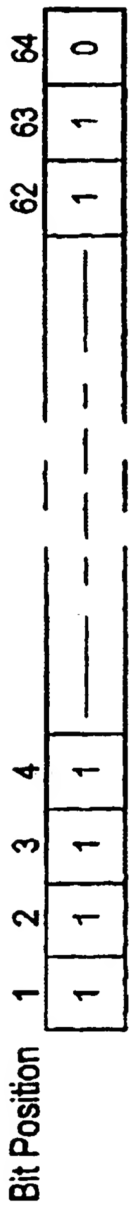


FIG. 11F

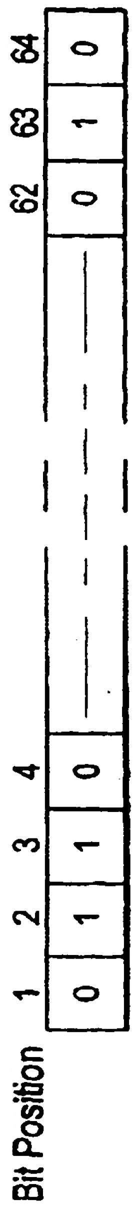
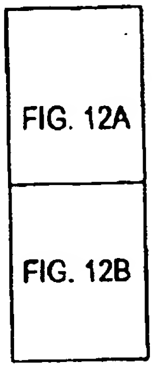


FIG. 11G

FIG. 12



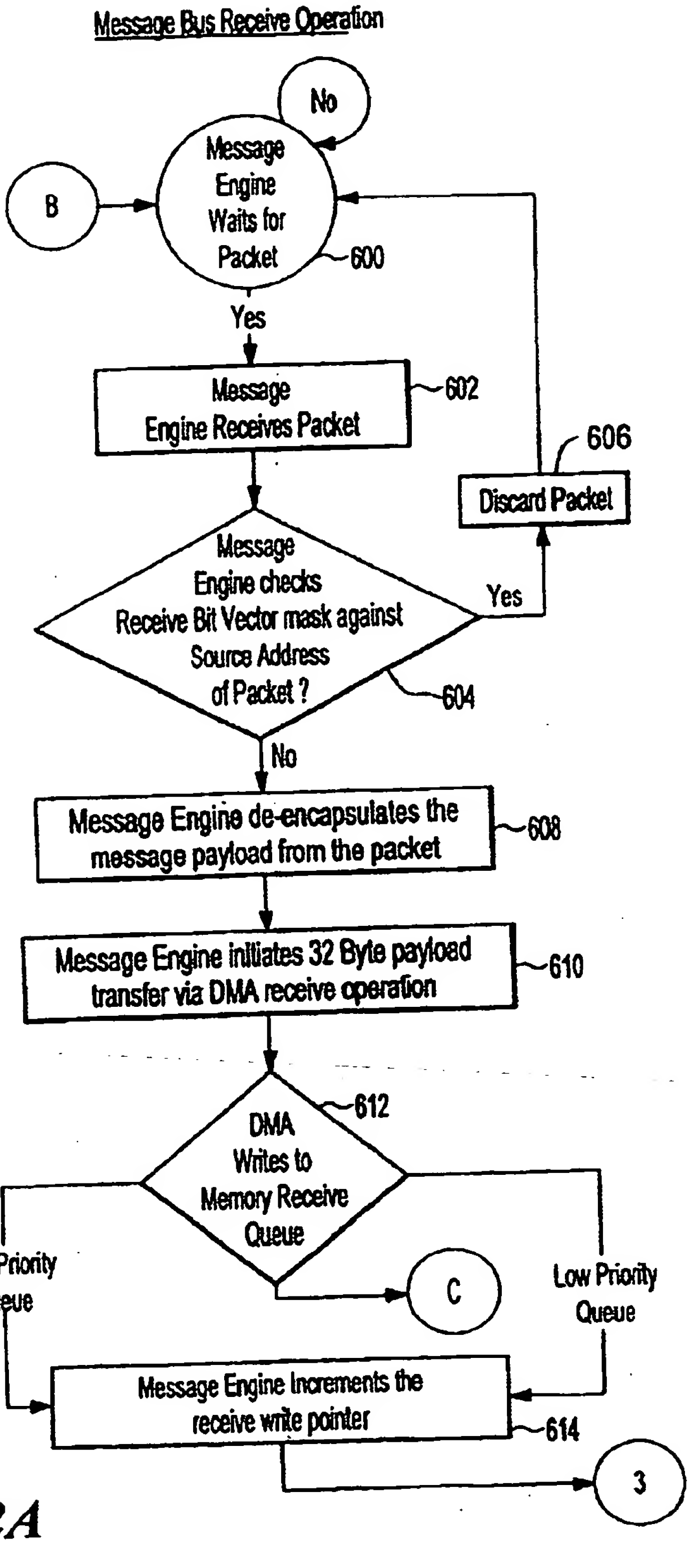

Message Bus Receive Operation
No
B
Message Engine Waits for Packet
600
Yes
Message Engine Receives Packet
602
606
Discard Packet
Message Engine checks Receive Bit Vector mask against Source Address of Packet ?
Yes
604
No
Message Engine de-encapsulates the message payload from the packet
608
Message Engine initiates 32 Byte payload transfer via DMA receive operation
610
612
DMA Writes to Memory Receive Queue
High Priority Queue
C
Low Priority Queue
Message Engine Increments the receive write pointer
614
3

FIG. 12A

Message Bus Receive Operation Continued

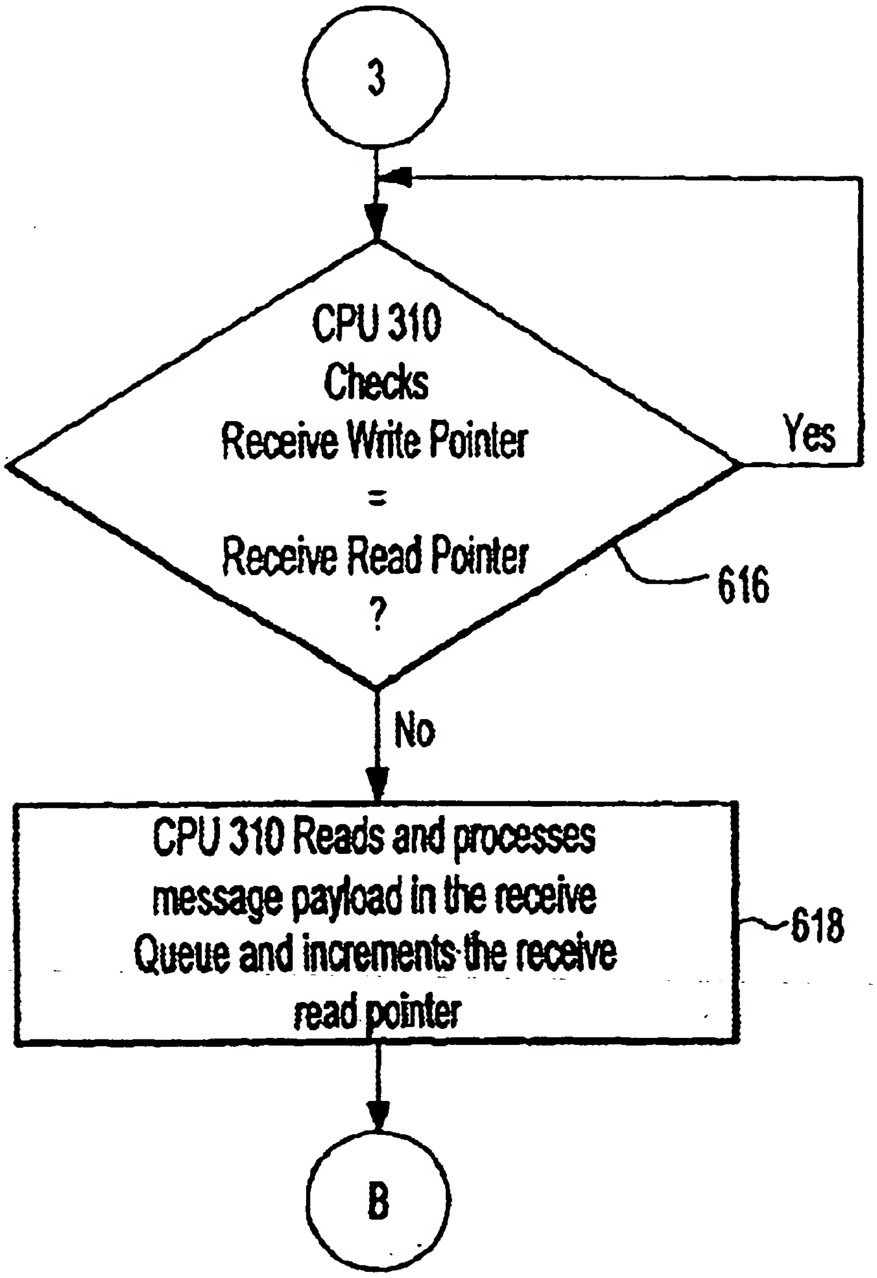

3
CPU 310 Checks Receive Write Pointer = Receive Read Pointer ?
Yes
616
No
CPU 310 Reads and processes message payload in the receive Queue and increments the receive read pointer
618
B

FIG 12B

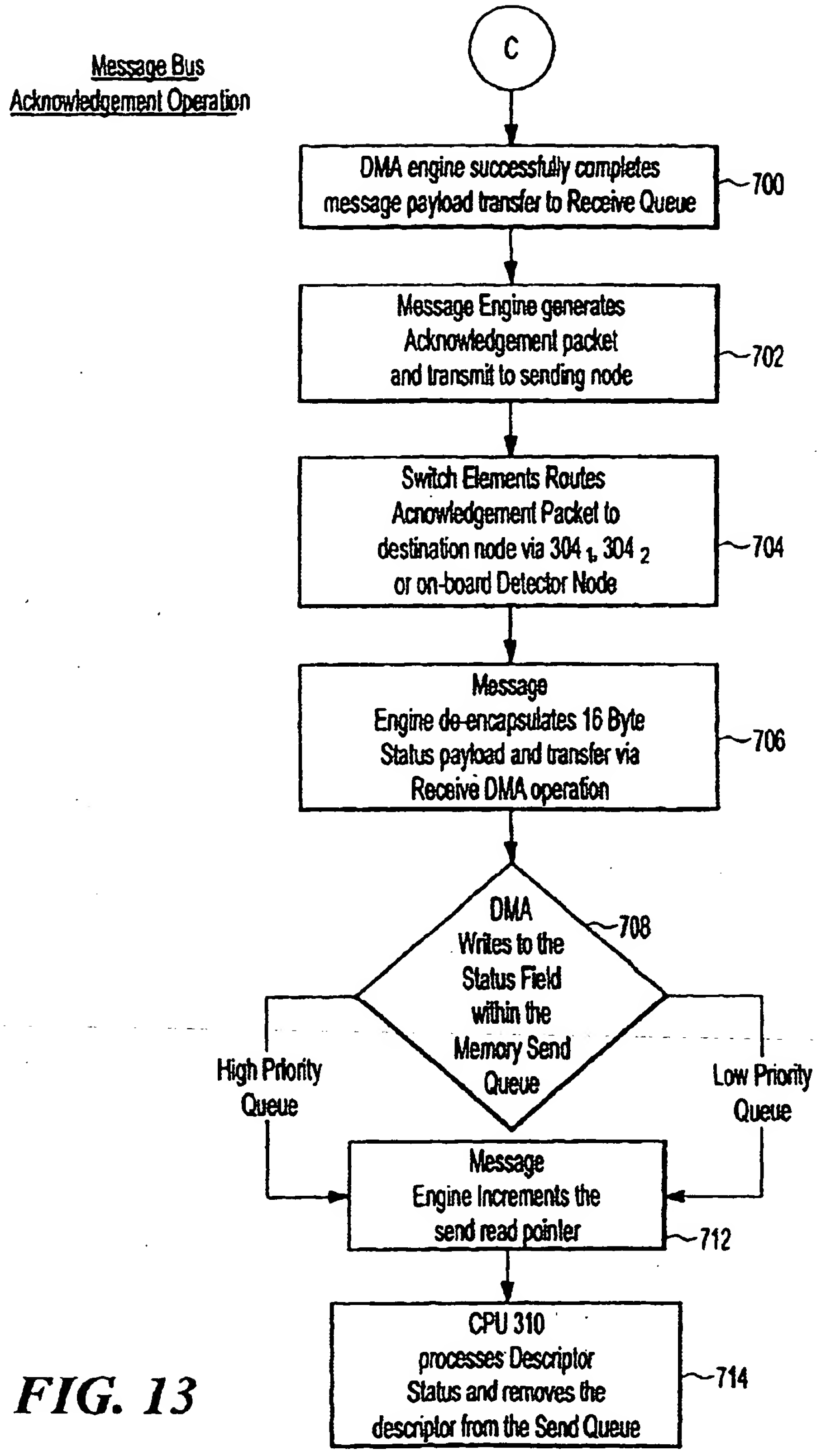

Message Bus
Acknowledgement Operation
C
DMA engine successfully completes message payload transfer to Receive Queue
700
Message Engine generates Acknowledgement packet and transmit to sending node
702
Switch Elements Routes Acnowledgement Packet to destination node via 304 1, 304 2 or on-board Detector Node
704
Message Engine de-encapsulates 16 Byte Status payload and transfer via Receive DMA operation
706
DMA Writes to the Status Field within the Memory Send Queue
708
High Priority Queue
Low Priority Queue
Message Engine Increments the send read pointer
712
CPU 310 processes Descriptor Status and removes the descriptor from the Send Queue
714

FIG. 13

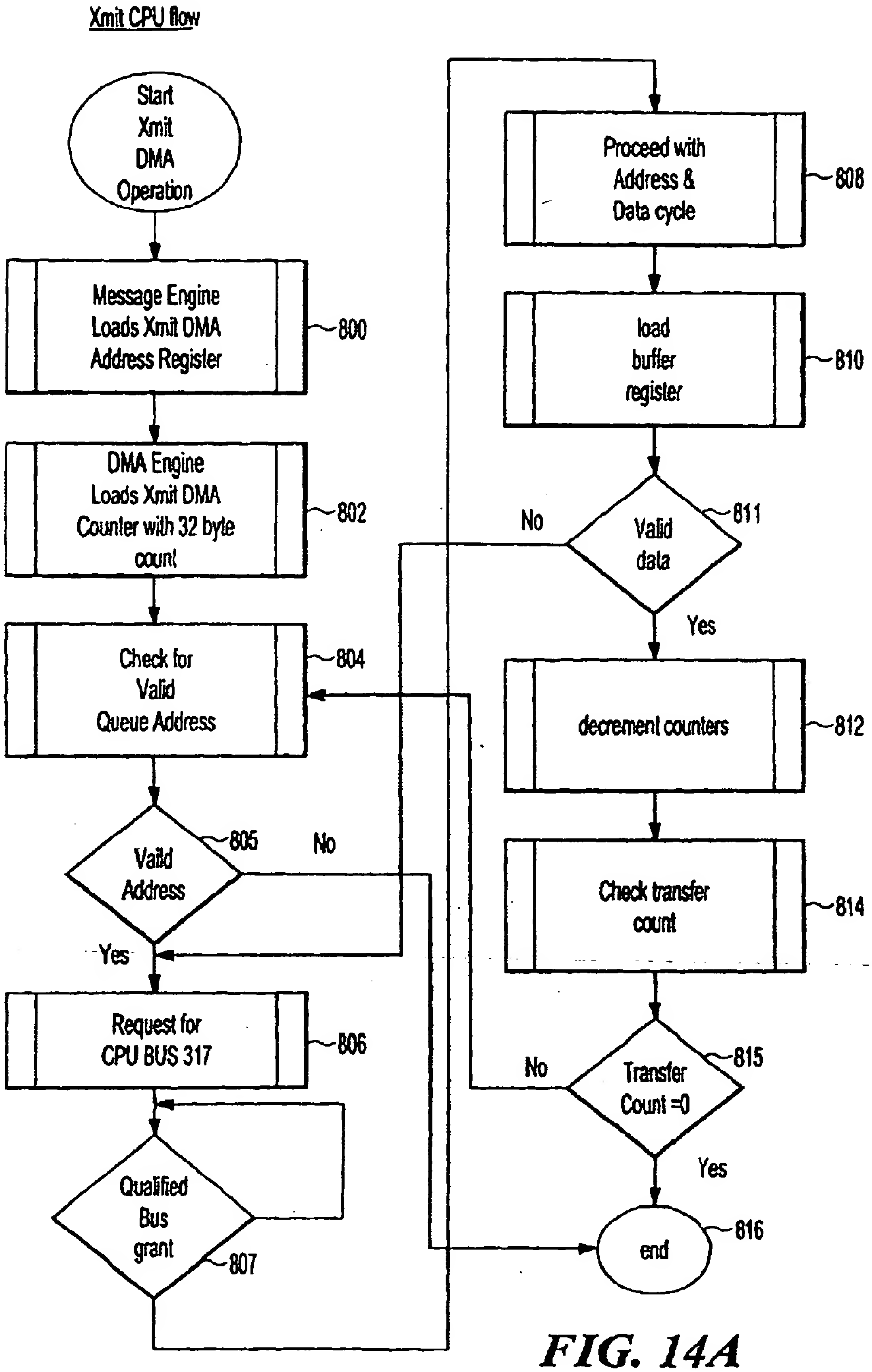

Xmit CPU flow
Start Xmit DMA Operation
Message Engine Loads Xmit DMA Address Register
800
DMA Engine Loads Xmit DMA Counter with 32 byte count
802
Check for Valid Queue Address
804
Vaild Address
805
No
Yes
Request for CPU BUS 317
806
Qualified Bus grant
807
Proceed with Address & Data cycle
808
load buffer register
810
Valid data
811
No
Yes
decrement counters
812
Check transfer count
814
Transfer Count =0
815
No
Yes
end
816

FIG. 14A

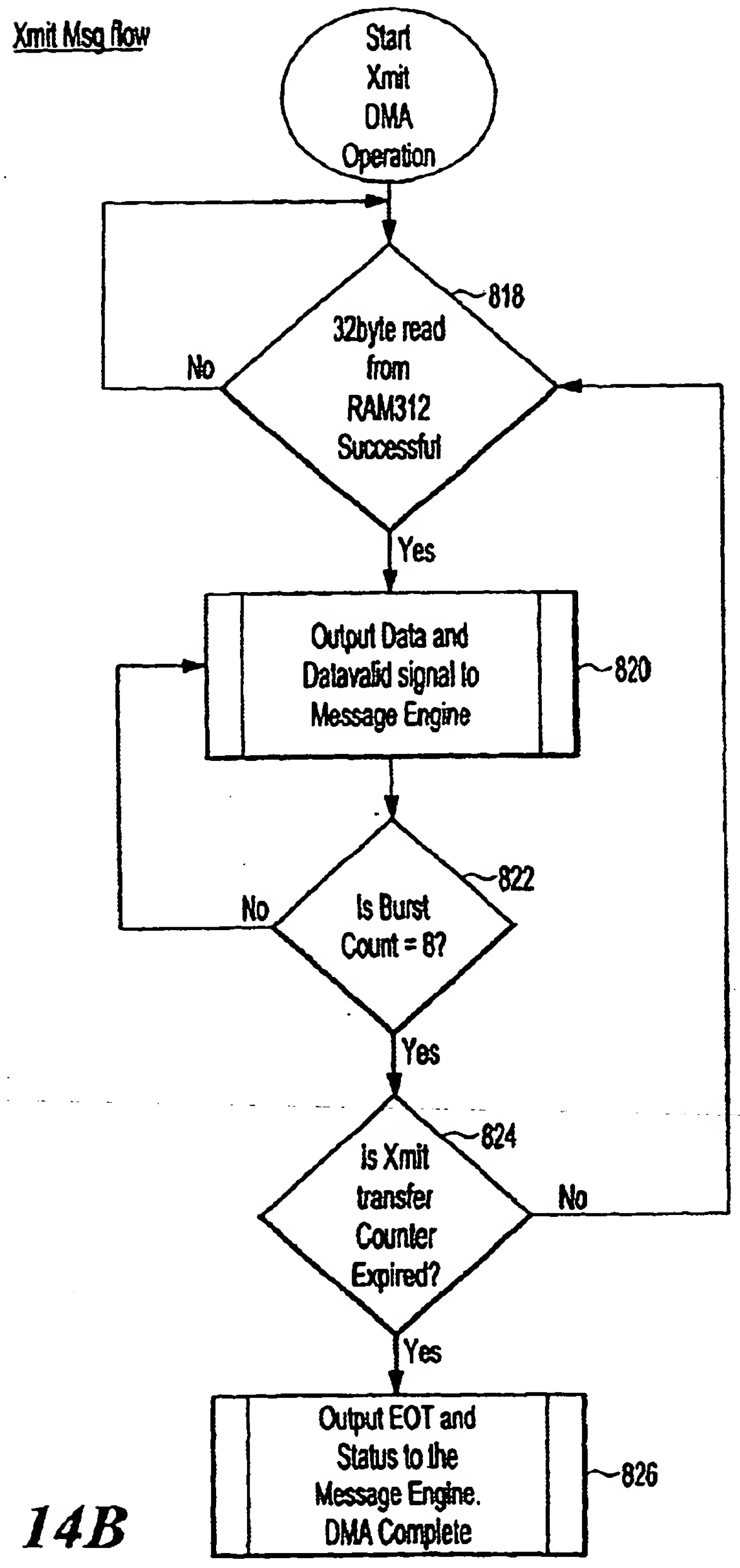

Xmit Msg flow
Start Xmit DMA Operation
818
32byte read from RAM312 Successful
No
Yes
Output Data and Datavalid signal to Message Engine
820
822
Is Burst Count = 8?
No
Yes
824
Is Xmit transfer Counter Expired?
No
Yes
Output EOT and Status to the Message Engine. DMA Complete
826

FIG. 14B

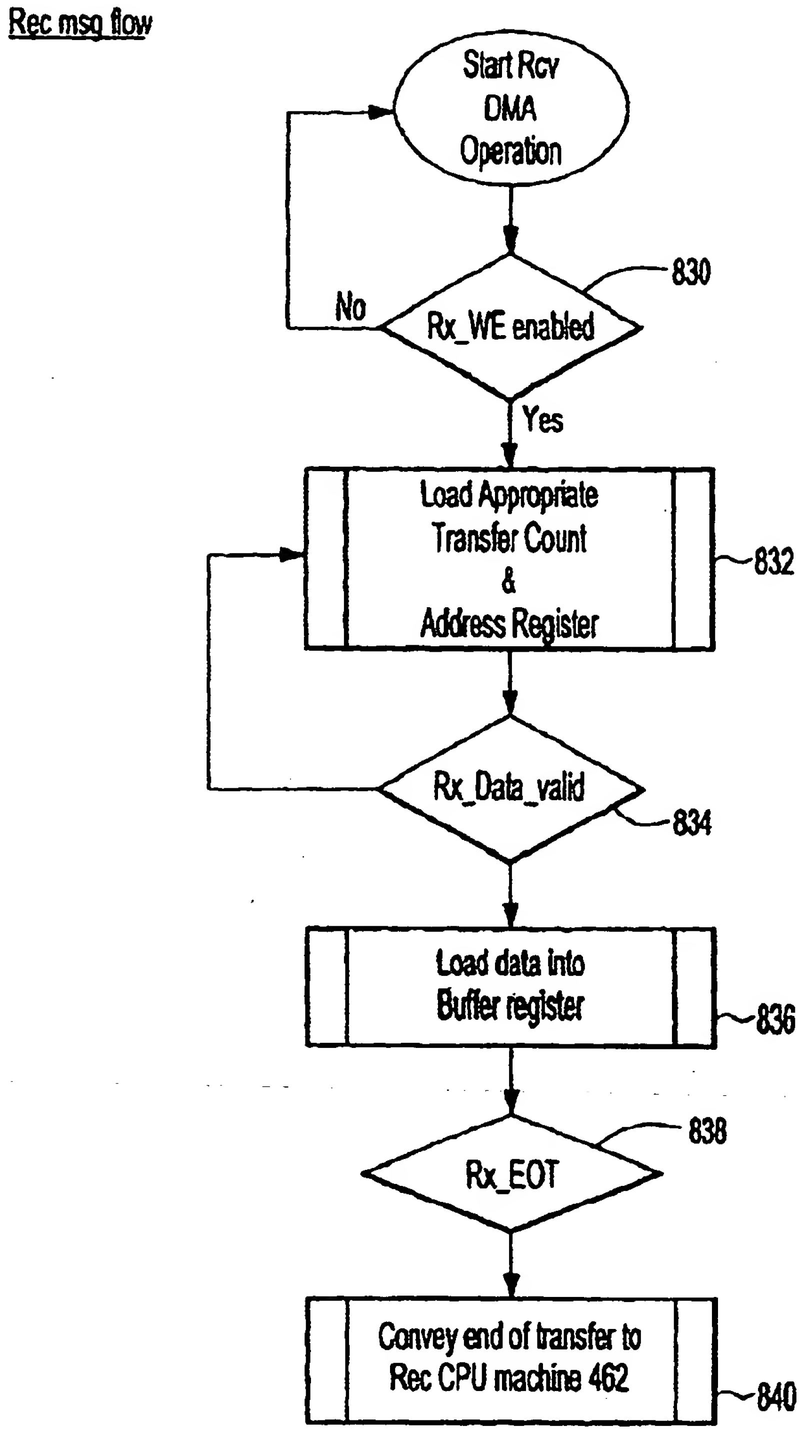

Rec msg flow
Start Rcv DMA Operation
Rx_WE enabled
830
No
Yes
Load Appropriate Transfer Count & Address Register
832
Rx_Data_valid
834
Load data into Buffer register
836
Rx_EOT
838
Convey end of transfer to Rec CPU machine 462
840

FIG. 15A

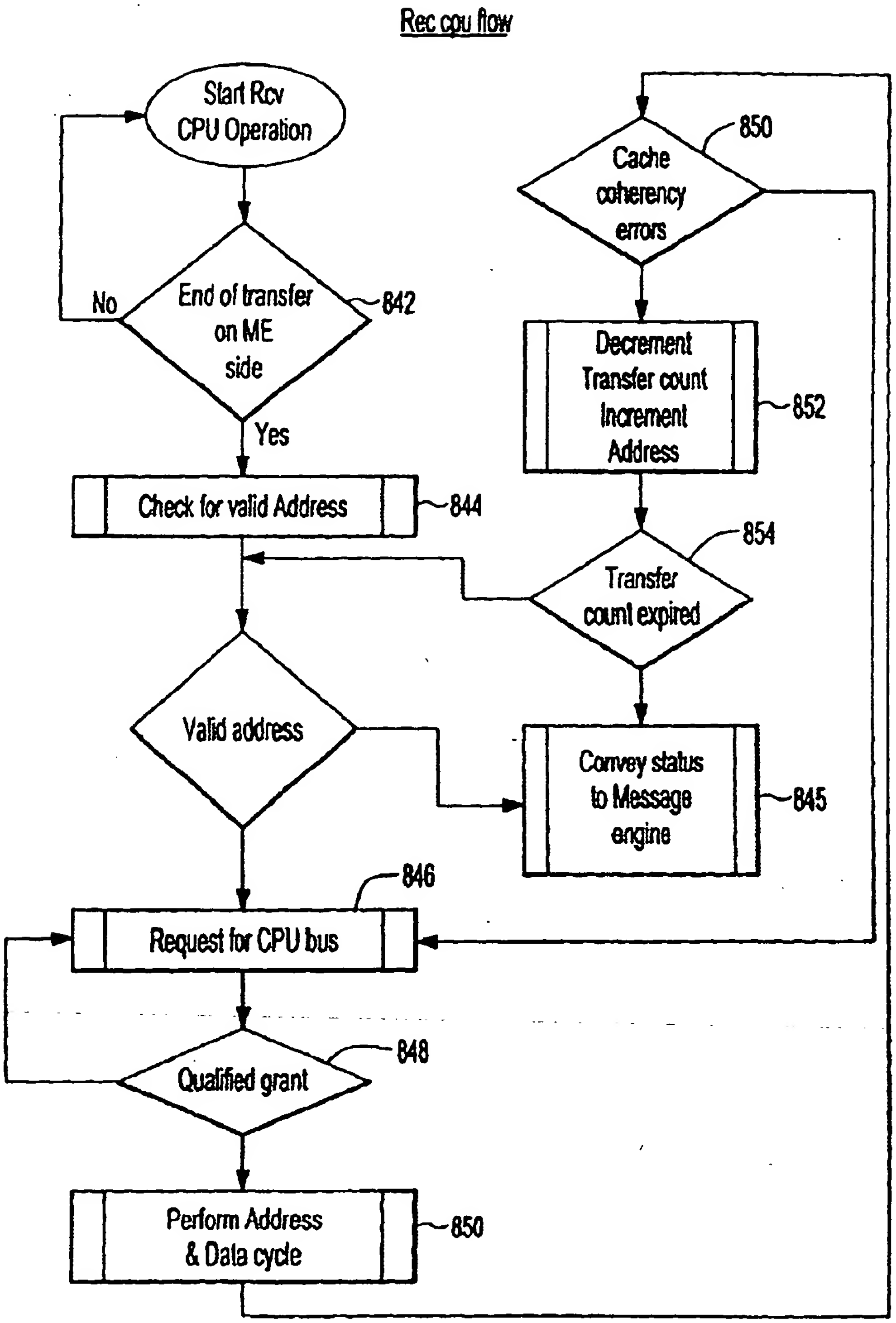

Rec cpu flow
Start Rcv CPU Operation
End of transfer on ME side
842
No
Yes
Check for valid Address
844
Valid address
Request for CPU bus
846
Qualified grant
848
Perform Address & Data cycle
850
Cache coherency errors
850
Decrement Transfer count Increment Address
852
Transfer count expired
854
Convey status to Message engine
845

FIG. 15B

DATA STORAGE SYSTEM HAVING SEPARATE DATA TRANSFER SECTION AND MESSAGE NETWORK WITH BUS ARBITRATION

BACKGROUND OF THE INVENTION

This invention relates generally to data storage systems, and more particularly to data storage systems having redundancy arrangements to protect against total system failure in the event of a failure in a component or subassembly of the storage system.

As is known in the art, large host computers and servers (collectively referred to herein as "host computer/servers") require large capacity data storage systems. These large computer/servers generally includes data processors, which perform many operations on data introduced to the host computer/server through peripherals including the data storage system. The results of these operations are output to peripherals, including the storage system.

One type of data storage system is a magnetic disk storage system. Here a bank of disk drives and the host computer/server are coupled together through an interface. The interface includes "front end" or host computer/server controllers (or directors) and "back-end" or disk controllers (or directors). The interface operates the controllers (or directors) in such a way that they are transparent to the host computer/server. That is, data is stored in, and retrieved from, the bank of disk drives in such a way that the host computer/server merely thinks it is operating with its own local disk drive. One such system is described in U.S. Pat. No. 5,206,939, entitled "System and Method for Disk Mapping and Data Retrieval", inventors Moshe Yanai, Natan Vishlitzky, Bruno Alterescu and Daniel Castel, issued Apr. 27, 1993, and assigned to the same assignee as the present invention.

As described in such U.S. Patent, the interface may also include, in addition to the host computer/server controllers (or directors) and disk controllers (or directors), addressable cache memories. The cache memory is a semiconductor memory and is provided to rapidly store data from the host computer/server before storage in the disk drives, and, on the other hand, store data from the disk drives prior to being sent to the host computer/server. The cache memory being a semiconductor memory, as distinguished from a magnetic memory as in the case of the disk drives, is much faster than the disk drives in reading and writing data.

The host computer/server controllers, disk controllers and cache memory are interconnected through a backplane printed circuit board. More particularly, disk controllers are mounted on disk controller printed circuit boards. The host computer/server controllers are mounted on host computer/server controller printed circuit boards. And, cache memories are mounted on cache memory printed circuit boards. The disk directors, host computer/server directors, and cache memory printed circuit boards plug into the backplane printed circuit board. In order to provide data integrity in case of a failure in a director, the backplane printed circuit board has a pair of buses. One set the disk directors is connected to one bus and another set of the disk directors is connected to the other bus. Likewise, one set the host computer/server directors is connected to one bus and another set of the host computer/server directors is directors connected to the other bus. The cache memories are connected to both buses. Each one of the buses provides data, address and control information.

The arrangement is shown schematically in FIG. 1. Thus, the use of two buses B1, B2 provides a degree of redundancy to protect against a total system failure in the event that the controllers or disk drives connected to one bus, fail. Further, the use of two buses increases the data transfer bandwidth of the system compared to a system having a single bus. Thus, in operation, when the host computer/server 12 wishes to store data, the host computer 12 issues a write request to one of the front-end directors 14 (i.e., host computer/server directors) to perform a write command. One of the front-end directors 14 replies to the request and asks the host computer 12 for the data. After the request has passed to the requesting one of the front-end directors 14, the director 14 determines the size of the data and reserves space in the cache memory 18 to store the request. The front-end director 14 then produces control signals on one of the address memory busses B1, B2 connected to such front-end director 14 to enable the transfer to the cache memory 18. The host computer/server 12 then transfers the data to the front-end director 14. The front-end director 14 then advises the host computer/server 12 that the transfer is complete. The front-end director 14 looks up in a Table, not shown, stored in the cache memory 18 to determine which one of the back-end directors 20 (i.e., disk directors) is to handle this request. The Table maps the host computer/server 12 addresses into an address in the bank 14 of disk drives. The front-end director 14 then puts a notification in a "mail box" (not shown and stored in the cache memory 18) for the back-end director 20, which is to handle the request, the amount of the data and the disk address for the data. Other back-end directors 20 poll the cache memory 18 when they are idle to check their "mail boxes". If the polled "mail box" indicates a transfer is to be made, the back-end director 20 processes the request, addresses the disk drive in the bank 22, reads the data from the cache memory 18 and writes it into the addresses of a disk drive in the bank 22.

When data is to be read from a disk drive in bank 22 to the host computer/server 12 the system operates in a reciprocal manner. More particularly, during a read operation, a read request is instituted by the host computer/server 12 for data at specified memory locations (i.e., a requested data block). One of the front-end directors 14 receives the read request and examines the cache memory 18 to determine whether the requested data block is stored in the cache memory 18. If the requested data block is in the cache memory 18, the requested data block is read from the cache memory 18 and is sent to the host computer/server 12. If the front-end director 14 determines that the requested data block is not in the cache memory 18 (i.e., a so-called "cache miss") and the director 14 writes a note in the cache memory 18 (i.e., the "mail box") that it needs to receive the requested data block. The back-end directors 20 poll the cache memory 18 to determine whether there is an action to be taken (i.e., a read operation of the requested block of data). The one of the back-end directors 20 which poll the cache memory 18 mail box and detects a read operation reads the requested data block and initiates storage of such requested data block stored in the cache memory 18. When the storage is completely written into the cache memory 18, a read complete indication is placed in the "mail box" in the cache memory 18. It is to be noted that the front-end directors 14 are polling the cache memory 18 for read complete indications. When one of the polling front-end directors 14 detects a read complete indication, such front-end director 14 completes the transfer of the requested data which is now stored in the cache memory 18 to the host computer/server 12.

The use of mailboxes and polling requires time to transfer data between the host computer/server 12 and the bank 22 of disk drives thus reducing the operating bandwidth of the interface.

SUMMARY OF THE INVENTION

In accordance with the present invention, a system interface is provided. Such interface includes a plurality of first directors, a plurality of second directors, a data transfer section and a message network. The data transfer section includes a cache memory. The cache memory is coupled to the plurality of first and second directors. The messaging network operates independently of the data transfer section and such network is coupled to the plurality of first directors and the plurality of second directors. The first and second directors control data transfer between the first directors and the second directors in response to message passing between the first directors and the second directors through the messaging network to facilitate data transfer between first directors and the second directors. The data passes through the cache memory in the data transfer section.

With such an arrangement, the cache memory in the data transfer section is not burdened with the task of transferring the director messaging but rather a messaging network is provided, operative independent of the data transfer section, for such messaging thereby increasing the operating bandwidth of the system interface.

In one embodiment of the invention, the system interface each one of the first directors includes a data pipe coupled between an input of such one of the first directors and the cache memory and a controller for transferring the messages between the message network and such one of the first directors.

In one embodiment each one of the second directors includes a data pipe coupled between an input of such one of the second directors and the cache memory and a controller for transferring the messages between the message network and such one of the second directors.

In one embodiment the directors include: a data pipe coupled between an input of such one of the first directors and the cache memory; a microprocessor; and a controller coupled to the microprocessor and the data pipe for controlling the transfer of the messages between the message network and such one of the first directors and for controlling the data between the input of such one of the first directors and the cache memory.

In accordance with another feature of the invention, a data storage system is provided for transferring data between a host computer/server and a bank of disk drives through a system interface. The system interface includes a plurality of first directors coupled to host computer/server, a plurality of second directors coupled to the bank of disk drives, a data transfer section, and a message network. The data transfer section includes a cache memory. The cache memory is coupled to the plurality of first and second directors. The message network is operative independently of the data transfer section and such network is coupled to the plurality of first directors and the plurality of second directors. The first and second directors control data transfer between the host computer and the bank of disk drives in response to messages passing between the first directors and the second directors through the messaging network to facilitate the data transfer between host computer/server and the bank of disk drives with such data passing through the cache memory in the data transfer section.

In accordance with yet another embodiment, a method is provided for operating a data storage system adapted to transfer data between a host computer/server and a bank of disk drives. The method includes transferring messages through a messaging network with the data being transferred between the host computer/server and the bank of disk drives through a cache memory, such message network being independent of the cache memory.

In accordance with another embodiment, a method is provided for operating a data storage system adapted to transfer data between a host computer/server and a bank of disk drives through a system interface. The interface includes a plurality of first directors coupled to host computer/server, a plurality of second directors coupled to the bank of disk drives; and a data transfer section having a cache memory, such cache memory being coupled to the plurality of first and second directors. The method comprises transferring the data between the host computer/server and the bank of disk drives under control of the first and second directors in response to messages passing between the first directors and the second directors through a messaging network to facilitate the data transfer between host computer/server and the bank of disk drives with such data passing through the cache memory in the data transfer section, such message network being independent of the cache memory.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the invention will become more readily apparent from the following detailed description when read together with the accompanying drawings, in which:

FIG. 1 is a block diagram of a data storage system according to the PRIOR ART;

FIG. 2 is a block diagram of a data storage system according to the invention;

FIG. 2A shows the fields of a descriptor used in the system interface of the data storage system of FIG. 2;

FIG. 2B shows the filed used in a MAC packet used in the system interface of the data storage system of FIG. 2;

FIG. 3 is a sketch of an electrical cabinet storing a system interface used in the data storage system of FIG. 2;

FIG. 4 is a diagramatical, isometric sketch showing printed circuit boards providing the system interface of the data storage system of FIG. 2;

FIG. 5 is a block diagram of the system interface used in the data storage system of FIG. 2;

FIG. 6 is a block diagram showing the connections between front-end and back-end directors to one of a pair of message network boards used in the system interface of the data storage system of FIG. 2;

FIG. 7 is a block diagram of an exemplary one of the director boards used in the system interface of he data storage system of FIG. 2;

FIG. 8 is a block diagram of the system interface used in the data storage system of FIG. 2;

FIG. 8A is a diagram of an exemplary global cache memory board used in the system interface of FIG. 8;

FIG. 8B is a diagram showing a pair of director boards coupled between a pair of host processors and global cache memory boards used in the system interface of FIG. 8;

FIG. 8C is a block diagram of an exemplary crossbar switch used in the front-end and read-end directors of the system interface of FIG. 8;

FIG. 9 is a block diagram of a transmit Direct Memory Access (DMA) used in the system interface of the FIG. 8;

FIG. 10 is a block diagram of a receive DMA used in the system interface of FIG. 8;

FIG. 11 shows the relationship between FIGS. 11A and 11B, such FIGS. 11A and 11B together showing a process flow diagram of the send operation of a message network used in the system interface of FIG. 8;

FIGS. 11C–11E are examples of digital words used by the message network in the system interface of FIG. 8;

FIG. 11F shows bits in a mask used in such message network,

FIG. 11G shows the result of the mask of FIG. 11F applied to the digital word shown in FIG. 11E;

FIG. 12 shows the relationship between FIGS. 12A and 12B, such FIGS. 12A and 12B Showing a process flow diagram of the receive operation of a message network used in the system interface of FIG. 8;

FIG. 13 shows the relationship between FIGS. 11A and 11B, such FIGS. 11A and 11B together showing a process flow diagram of the acknowledgement operation of a message network used in the system interface of FIG. 8;

FIGS. 14A and 14B show process flow diagrams of the transmit DMA operation of the transmit DMA of FIG. 9; and

FIGS. 15A and 15B show process flow diagrams of the receive DMA operation of the receive DMA of FIG. 10.

DETAILED DESCRIPTION

Referring now to FIG. 2, a data storage system 100 is shown for transferring data between a host computer/server 120 and a bank of disk drives 140 through a system interface 160. The system interface 160 includes: a plurality of, here 32 front-end directors 180_1–180_{32} coupled to the host computer/server 120 via ports-123_{32}; a plurality of back-end directors 200_1–200_{32} coupled to the bank of disk drives 140 via ports 123_{33}–123_{64}; a data transfer section 240, having a global cache memory 220, coupled to the plurality of front-end directors 180_1–180_{16} and the back-end directors 200_1–200_{16}; and a messaging network 260, operative independently of the data transfer section 240, coupled to the plurality of front-end directors 180_1–180_{32} and the plurality of back-end directors 200_1–200_{32}, as shown. The front-end and back-end directors 180_1–180_{32}, 200_1–200_{32} are functionally similar and include a microprocessor (μP) 299 (i.e., a central processing unit (CPU) and RAM), a message engine/CPU controller 314 and a data pipe 316 to be described in detail in connection with FIGS. 5, 6 and 7. Suffice it to say here, however, that the front-end and back-end directors 180_1–180_{32}, 200_1–200_{32} control data transfer between the host computer/server 120 and the bank of disk drives 140 in response to messages passing between the directors 180_1–180_{32}, 200_1–200_{32} through the messaging network 260. The messages facilitate the data transfer between host computer/server 120 and the bank of disk drives 140 with such data passing through the global cache memory 220 via the data transfer section 240. More particularly, in the case of the front-end directors 180_1–180_{32}, the data passes between the host computer to the global cache memory 220 through the data pipe 316 in the front-end directors 180_1–180_{32} and the messages pass through the message engine/CPU controller 314 in such front-end directors 180_1–180_{32}. In the case of the back-end directors 200_1–200_{32} the data passes between the back-end directors 200_1–200_{32} and the bank of disk drives 140 and the global cache memory 220 through the data pipe 316 in the back-end directors 200_1–200_{32} and again the messages pass through the message engine/CPU controller 314 in such back-end director 200_1–200_{32}.

With such an arrangement, the cache memory 220 in the data transfer section 240 is not burdened with the task of transferring the director messaging. Rather the messaging

network 260 operates independent of the data transfer section 240 thereby increasing the operating bandwidth of the system interface 160.

In operation, and considering first a read request by the host computer/server 120 (i.e., the host computer/server 120 requests data from the bank of disk drives 140), the request is passed from one of a plurality of, here 32, host computer processors 121_1–121_{32} in the host computer 120 to one or more of the pair of the front-end directors 180_1–180_{32} connected to such host computer processor 121_1–121_{32}. (It is noted that in the host computer 120, each one of the host computer processors 121_1–121_{32} is coupled to here a pair (but not limited to a pair) of the front-end directors 180_1–180_{32}, to provide redundancy in the event of a failure in one of the front end-directors 180_1–180_{32} coupled thereto. Likewise, the bank of disk drives 140 has a plurality of, here 32, disk drives 141_1–141_{32}, each disk drive 141_1–141_{32} being coupled to here a pair (but not limited to a pair) of the back-end directors 200_1–200_{32}, to provide redundancy in the event of a failure in one of the back-end directors 200_1–200_{32} coupled thereto). Each front-end director 180_1–180_{32} includes a microprocessor (μP) 299 (i.e., a central processing unit (CPU) and RAM) and will be described in detail in connection with FIGS. 5 and 7. Suffice it to say here, however, that the microprocessor 299 makes a request from the data from the global cache memory 220. The global cache memory 220 has a resident cache management table, not shown. Every director 180_1–180_{32}, 200_1–200_{32} has access to the resident cache management table and every time a front-end director 180_1–180_{32} requests a data transfer, the front-end director 180_1–180_{32} must query the global cache memory 220 to determine whether the requested data is in the global cache memory 220. If the requested data is in the global cache memory 220 (i.e., a read "hit"), the front-end director 180_1–180_{32}, more particularly the microprocessor 299 therein, mediates a DMA (Direct Memory Access) operation for the global cache memory 220 and the requested data is transferred to the requesting host computer processor 121_1–121_{32}.

If, on the other hand, the front-end director 180_1–180_{32} receiving the data request determines that the requested data is not in the global cache memory 220 (i.e., a "miss") as a result of a query of the cache management table in the global cache memory 220, such front-end director 180_1–180_{32} concludes that the requested data is in the bank of disk drives 140. Thus the front-end director 180_1–180_{32} that received the request for the data must make a request for the data from one of the back-end directors 200_1–200_{32} in order for such back-end director 200_1–200_{32} to request the data from the bank of disk drives 140. The mapping of which back-end directors 200_1–200_{32} control which disk drives 141_1–141_{32} in the bank of disk drives 140 is determined during a power-up initialization phase. The map is stored in the global cache memory 220. Thus, when the front-end director 180_1–180_{32} makes a request for data from the global cache memory 220 and determines that the requested data is not in the global cache memory 220 (i.e., a "miss"), the front-end director 180_1–180_{32} is also advised by the map in the global cache memory 220 of the back-end director 200_1–200_{32} responsible for the requested data in the bank of disk drives 140. The requesting front-end director 180_1–180_{32} then must make a request for the data in the bank of disk drives 140 from the map designated back-end director 200_1–200_{32}. This request between the front-end director 180_1–180_{32} and the appropriate one of the back-end directors 200_1–200_{32} (as determined by the map stored in the global cache memory 200) is by a message which passes from the front-end director 180_1–180_{32} through the message network 260 to the appropriate back-end director 200_1–200_{32}. It is noted then that the message does not pass through the global cache memory 220 (i.e., does not pass through the data transfer section 240) but rather passes through the separate, independent message network 260. Thus, communication between the directors 180_1–180_{32}, 200_1–200_{32} is through the message network 260 and not through the global cache memory 220. Consequently, valuable bandwidth for the global cache memory 220 is not used for messaging among the directors 180_1–180_{32}, 200_1–200_{32}.

Thus, on a global cache memory 220 "read miss", the front-end director 180_1–180_{32} sends a message to the appropriate one of the back-end directors 200_1–200_{32} through the message network 260 to instruct such back-end director 200_1–200_{32} to transfer the requested data from the bank of disk drives 140 to the global cache memory 220. When accomplished, the back-end director 200_1–200_{32} advises the requesting front-end director 180_1–180_{32} that the transfer is accomplished by a message, which passes from the back-end director 200_1–200_{32} to the front-end director 180_1–180_{32} through the message network 260. In response to the acknowledgement signal, the front-end director 180_1–180_{32} is thereby advised that such front-end director 180_1–180_{32} can transfer the data from the global cache memory 220 to the requesting host computer processor 121_1–121_{32} as described above when there is a cache "read hit".

It should be noted that there might be one or more back-end directors 200_1–200_{32} responsible for the requested data. Thus, if only one back-end director 200_1–200_{32} is responsible for the requested data, the requesting front-end director 180_1–180_{32} sends a uni-cast message via the message network 260 to only that specific one of the back-end directors 200_1–200_{32}. On the other hand, if more than one of the back-end directors 200_1–200_{32} is responsible for the requested data, a multi-cast message (here implemented as a series of uni-cast messages) is sent by the requesting one of the front-end directors 180_1–180_{32} to all of the back-end directors 200_1–200_{32} having responsibility for the requested data. In any event, with both a uni-cast or multi-cast message, such message is passed through the message network 260 and not through the data transfer section 240 (i.e., not through the global cache memory 220).

Likewise, it should be noted that while one of the host computer processors 121_1–121_{32} might request data, the acknowledgement signal may be sent to the requesting host computer processor 121_1 or one or more other host computer processors 121_1–121_{32} via a multi-cast (i.e., sequence of uni-cast) messages through the message network 260 to complete the data read operation.

Considering a write operation, the host computer 120 wishes to write data into storage (i.e., into the bank of disk drives 140). One of the front-end directors 180_1–180_{32} receives the data from the host computer 120 and writes it into the global cache memory 220. The front-end director 180_1–180_{32} then requests the transfer of such data after some period of time when the back-end director 200_1–200_{32} determines that the data can be removed from such cache memory 220 and stored in the bank of disk drives 140. Before the transfer to the bank of disk drives 140, the data in the cache memory 220 is tagged with a bit as "fresh data" (i.e., data which has not been transferred to the bank of disk drives 140, that is data which is "write pending"). Thus, if there are multiple write requests for the same memory location in the global cache memory 220 (e.g., a particular bank account) before being transferred to the bank of disk drives 140, the data is overwritten in the cache memory 220 with the most recent data. Each time data is transferred to the global cache memory 220, the front-end director 180_1–180_{32} controlling the transfer also informs the host computer 120 that the transfer is complete to thereby free-up the host computer 120 for other data transfers.

When it is time to transfer the data in the global cache memory 220 to the bank of disk drives 140, as determined by the back-end director 200_1–200_{32}, the back-end director 200_1–200_{32} transfers the data from the global cache memory 220 to the bank of disk drives 140 and resets the tag associated with data in the global cache memory 220 (i.e., un-tags the data) to indicate that the data in the global cache memory 220 has been transferred to the bank of disk drives 140. It is noted that the un-tagged data in the global cache memory 220 remains there until overwritten with new data.

Referring now to FIGS. 3 and 4, the system interface 160 is shown to include an electrical cabinet 300 having stored therein: a plurality of, here eight front-end director boards 190_1–190_8, each one having here four of the front-end directors 180_1–180_{32}; a plurality of, here eight back-end director boards 210_1–210_8, each one having here four of the back-end directors 200_1–200_{32}; and a plurality of, here eight, memory boards 220' which together make up the global cache memory 220. These boards plug into the front side of a backplane 302. (It is noted that the backplane 302 is a mid-plane printed circuit board). Plugged into the backside of the backplane 302 are message network boards 304_1, 304_2. The backside of the backplane 302 has plugged into it adapter boards, not shown in FIGS. 2–4, which couple the boards plugged into the back-side of the backplane 302 with the computer 120 and the bank of disk drives 140 as shown in FIG. 2. That is, referring again briefly to FIG. 2, an I/O adapter, not shown, is coupled between each one of the front-end directors 180_1–180_{32} and the host computer 120 and an I/O adapter, not shown, is coupled between each one of the back-end directors 200_1–200_{32} and the bank of disk drives 140.

Referring now to FIG. 5, the system interface 160 is shown to include the director boards 190_1–190_8, 210_1–210_8 and the global cache memory 220, plugged into the backplane 302 and the disk drives 141_1–141_{32} in the bank of disk drives along with the host computer 120 also plugged into the backplane 302 via I/O adapter boards, not shown. The message network 260 (FIG. 2) includes the message network boards 304_1 and 304_2. Each one of the message network boards 304_1 and 304_2 is identical in construction. A pair of message network boards 304_1 and 304_2 is used for redundancy and for message load balancing. Thus, each message network board 304_1, 304_2, includes a controller 306 (i.e., an initialization and diagnostic processor comprising a CPU, system controller interface and memory, as shown in FIG. 6 for one of the message network boards 304_1, 304_2, here board 304_1) and a crossbar switch section 308 (e.g., a switching fabric made up of here four switches 308_1–308_4).

Referring again to FIG. 5, each one of the director boards 190_1–210_8 includes, as noted above four of the directors 180_1–180_{32}, 200_1–200_{32} (FIG. 2). It is noted that the director boards 190_1–190_8 having four front-end directors per board, 180_1–180_{32} are referred to as front-end directors and the director boards 210_1–210_8 having four back-end directors per board, 200_1–200_{32} are referred to as back-end directors. Each one of the directors 180_1–180_{32}, 200_1–200_{32} includes a CPU 310, a RAM 312 (which make up the microprocessor 299 referred to above), the message engine/CPU controller 314, and the data pipe 316.

Each one of the director boards 190_1–210_8 includes a crossbar switch 318. The crossbar switch 318 has four input/output ports 319, each one being coupled to the data pipe 316 of a corresponding one of the four directors 180_1–180_{32}, 200_1–200_{32} on the director board.190_1–210_8. The crossbar switch 318 has eight output/input ports collectively identified in FIG. 5 by numerical designation 321 (which plug into the backplane 302. The crossbar switch 318 on the front-end director boards 191_1–191_8 is used for coupling the data pipe 316 of a selected one of the four front-end directors 180_1–180_{32} on the front-end director board 190_1–190_8 to the global cache memory 220 via the backplane 302 and I/O adapter, not shown. The crossbar switch 318 on the back-end director boards 210_1–210_8 is used for coupling the data pipe 316 of a selected one of the four back-end directors 200_1–200_{32} on the back-end director board 210_1–210_8 to the global cache memory 220 via the backplane 302 and I/O adapter, not shown. Thus, referring to FIG. 2, the data pipe 316 in the front-end directors 180_1–180_{32} couples data between the host computer 120 and the global cache memory 220 while the data pipe 316 in the back-end directors 200_1–200_{32} couples data between the bank of disk drives 140 and the global cache memory 220. It is noted that there are separate point-to-point data paths P_1–P_{64} (FIG. 2) between each one of the directors 180_1–180_{32}, 200_1–200_{32} and the global cache memory 220. It is also noted that the backplane 302 is a passive backplane because it is made up of only etched conductors on one or more layers of a printed circuit board. That is, the backplane 302 does not have any active components.

Referring again to FIG. 5, each one of the director boards 190_1–210_8 includes a crossbar switch 320. Each crossbar switch 320 has four input/output ports 323, each one of the four input/output ports 323 being coupled to the message engine/CPU controller 314 of a corresponding one of the four directors 180_1–180_{32}, 200_1–200_{32} on the director board 190_1–210_8. Each crossbar switch 320 has a pair of output/input ports 325_1, 325_2, which plug into the backplane 302. Each port 325_1–325_2 is coupled to a corresponding one of the message network boards 304_1, 304_2, respectively, through the backplane 302. The crossbar switch 320 on the front-end director boards 190_1–190_8 is used to couple the messages between the message engine/CPUcontroller 314 of a selected one of the four front-end directors 180_1–180_{32} on the front-end director boards 190_1–190_8 and the message network 260, FIG. 2. Likewise, the back-end director boards 210_1–210_8 are used to couple the messages produced by a selected one of the four back-end directors 200_1–200_{32} on the back-end director board 210_1–210_8 between the message engine/CPU controller 314 of a selected one of such four back-end directors and the message network 260 (FIG. 2). Thus, referring also to FIG. 2, instead of having a separate dedicated message path between each one of the directors 180_1–180_{32}, 200_1–200_{32} and the message network 260 (which would require M individual connections to the backplane 302 for each of the directors, where M is an integer), here only M/4 individual connections are required). Thus, the total number of connections between the directors 180_1–180_{32}, 200_1–200_{32} and the backplane 302 is reduced to ¼th. Thus, it should be noted from FIGS. 2 and 5 that the message network 260 (FIG. 2) includes the crossbar switch 320 and the message network boards 304_1, 304_2.

Each message is a 64-byte descriptor, shown in FIG. 2A) which is created by the CPU 310 (FIG. 5) under software control and is stored in a send queue in RAM 312. When the message is to be read from the send queue in RAM 312 and transmitted through the message network 260 (FIG. 2) to one or more other directors via a DMA operation to be described, it is packetized in the packetizer portion of

packetizer/de-packetizer 428 (FIG. 7) into a MAC type packet, shown in FIG. 2B, here using the NGIO protocol specification. There are three types of packets: a message packet section; an acknowledgement packet; and a message network fabric management packet, the latter being used to establish the message network routing during initialization (i.e., during power-up). Each one of the MAC packets has: an 8-byte header which includes source (i.e., transmitting director) and destination (i.e., receiving director) address; a payload; and terminates with a 4-byte Cyclic Redundancy Check (CRC), as shown in FIG. 2B. The acknowledgement packet (i.e., signal) has a 4-byte acknowledgment payload section. The message packet has a 32-byte payload section. The Fabric Management Packet (FMP) has a 256-byte payload section. The MAC packet is sent to the crossbar switch 320. The destination portion of the packet is used to indicate the destination for the message and is decoded by the switch 320 to determine which port the message is to be routed. The decoding process uses a decoder table 327 in the switch 318, such table being initialized by controller during power-up by the initialization and diagnostic processor (controller) 306 (FIG. 5). The table 327 (FIG. 7) provides the relationship between the destination address portion of the MAC packet, which identifies the routing for the message and the one of the four directors 180_1–180_{32}, 200_1–200_{32} on the director board 190_1–190_8, 210_1–210_8 or to one of the message network boards 304_1, 304_2 to which the message is to be directed.

More particularly, and referring to FIG. 5, a pair of output/input ports 325_1, 325_2 is provided for each one of the crossbar switches 320, each one being coupled to a corresponding one of the pair of message network boards 304_1, 304_2. Thus, each one of the message network boards 304_1, 304_2 has sixteen input/output ports 322_1–322_{16}, each one being coupled to a corresponding one of the output/input ports 325_1, 325_2, respectively, of a corresponding one of the director boards 190_1–190_8, 210_1–210_8 through the backplane 302, as shown. Thus, considering exemplary message network board 304_1, FIG. 6, each switch 308_1–308_4 also includes three coupling ports 324_1–324_3. The coupling ports 324_1–324_3 are used to interconnect the switches 322_1–322_4, as shown in FIG. 6. Thus, considering message network board 304_1, input/output ports 322_1–322_8 are coupled to output/input ports 325_1 of front-end director boards 190_1–190_8 and input/output ports 322_9–322_{16} are coupled to output/input ports 325_1 of back-end director boards 210_1–210_8, as shown. Likewise, considering message network board 304_2, input/output ports 322_1–322_8 thereof are coupled, via the backplane 302, to output/input ports 325_2 of front-end director boards 190_1–190_8 and input/output ports 322_9–322_{16} are coupled, via the backplane 302, to output/input ports 325_2 of back-end director boards 210_1–210_8.

As noted above, each one of the message network boards 304_1, 304_2 includes a processor 306 (FIG. 5) and a crossbar switch section 308 having four switches 308_1–308_4, as shown in FIGS. 5 and 6. The switches 308_1–308_4 are interconnected as shown so that messages can pass between any pair of the input/output ports 322_1–322_{16}. Thus, it follow that a message from any one of the front-end directors 180_1–180_{32} can be coupled to another one of the front-end directors 180_1.180_{32} and/or to any one of the back-end directors 200_1–200_{32}. Likewise, a message from any one of the back-end directors 180_1–180_{32} can be coupled to another one of the back-end directors 180_1.180_{32} and/or to any one of the front-end directors 200_1–200_{32}.

As noted above, each MAC packet (FIG. 2B) includes in an address destination portion and a data payload portion.

The MAC header is used to indicate the destination for the MAC packet and such MAC header is decoded by the switch to determine which port the MAC packet is to be routed. The decoding process uses a table in the switch 308_1–308_4, such table being initialized by processor 306 during power-up. The table provides the relationship between the MAC header, which identifies the destination for the MAC packet and the route to be taken through the message network. Thus, after initialization, the switches 320 and the switches 308_1–308_4 in switch section 308 provides packet routing which enables each one of the directors 180_1–180_{32}, 200_1–200_{32} to transmit a message between itself and any other one of the directors, regardless of whether such other director is on the same director board 190_1–180_8, 210_1–210_8 or on a different director board. Further, the MAC packet has an additional bit B in the header thereof, as shown in FIG. 2B, which enables the message to pass through message network board 304_1 or through message network board 304_2. During normal operation, this additional bit B is toggled between a logic 1 and a logic 0 so that one message passes through one of the redundant message network boards 304_1, 304_2 and the next message to pass through the one of the message network boards 304_1, 304_2 to balance the load requirement on the system. However, in the event of a failure in one of the message network boards 304_1, 304_2, the non-failed one of the boards 304_1, 304_2 is used exclusively until the failed message network board is replaced.

Referring now to FIG. 7, an exemplary one of the director boards 190_1–190_8, 210_1–210_8, here director board 190_1 is shown to include directors 180_1, 180_3, 180_5 and 180_7. An exemplary one of the directors 180_1–180_4, here 180_1 is shown in detail to include the data pipe 316, the message engine/CPU controller 314, the RAM 312, and the CPU 310 all coupled to the CPU interface bus 317, as shown. The exemplary director 180_1 also includes: a local cache memory 319 (which is coupled to the CPU 310); the crossbar switch 318; and, the crossbar switch 320, described briefly above in connection with FIGS. 5 and 6. The data pipe 316 includes a protocol translator 400, a quad port RAM 402 and a quad port RAM controller 404 arranged as shown. Briefly, the protocol translator 400 converts between the protocol of the host computer 120, in the case of a front-end director 180_1–180_{32}, (and between the protocol used by the disk drives in bank 140 in the case of a back-end director 200_1–200_{32}) and the protocol between the directors 180_1–180_3, 200_1–200_{32} and the global memory 220 (FIG. 2). More particularly, the protocol used the host computer 120 may, for example, be fibre channel, SCSI, ESCON or FICON, for example, as determined by the manufacture of the host computer 120 while the protocol used internal to the system interface 160 (FIG. 2) may be selected by the manufacturer of the interface 160. The quad port RAM 402 is a FIFO controlled by controller 404 because the rate data coming into the RAM 402 may be different from the rate data leaving the RAM 402. The RAM 402 has four ports, each adapted to handle an 18 bit digital word. Here, the protocol translator 400 produces 36 bit digital words for the system interface 160 (FIG. 2) protocol, one 18 bit portion of the word is coupled to one of a pair of the ports of the quad port RAM 402 and the other 18 bit portion of the word is coupled to the other one of the pair of the ports of the quad port RAM 402. The quad port RAM has a pair of ports 402A, 402B, each one of to ports 402A, 402B being adapted to handle an 18 bit digital word. Each one of the ports 402A, 402B is independently controllable and has independent, but arbitrated, access to the memory array within the RAM 402. Data is transferred between the ports 402A, 402B and the cache memory 220 (FIG. 2) through the crossbar switch 318, as shown.

The crossbar switch 318 includes a pair of switches 406A, 406B. Each one of the switches 406A, 406B includes four input/output director-side ports D_1–D_4 (collectively referred to above in connection with FIG. 5 as port 319) and four input/output memory-side ports M_1–M_4, M_5–M_8, respectively, as indicated. The input/output memory-side ports M_1–M_4, M_5–M_8 were collectively referred to above in connection with FIG. 5 as port 317). The director-side ports D_1–D_4 of switch 406A are connected to the 402A ports of the quad port RAMs 402 in each one the directors 180_1, 180_3, 180_5 and 180_7, as indicated. Likewise, director-side ports of switch 406B are connected to the 402B ports of the quad port RAMs 402 in each one the directors 180_1, 180_3, 180_5, and 180_7, as indicated. The ports D_1–D_4 are selectively coupled to the ports M_1–M_4 in accordance with control words provided to the switch 406A by the controllers in directors 180_1, 180_3, 180_5, 180_7 on busses R_{A1}–R_{A4}, respectively, and the ports D_1–D_4 are coupled to ports M_5–M_8 in accordance with the control words provided to switch 406B by the controllers in directors 180_1, 180_3, 180_5, 180_7 on busses R_{B1}–R_{B4}, as indicated. The signals on buses R_{A1}–R_{A4} are request signals. Thus, port 402A of any one of the directors 180_1, 180_3, 180_5, 180_7 may be coupled to any one of the ports M_1–M_4 of switch 406A, selectively in accordance with the request signals on buses R_{A1}–R_{A4}. Likewise, port 402B of any one of the directors 180_1–180_4 may be coupled to any one of the ports M_5–M_8 of switch 406B, selectively in accordance with the request signals on buses R_{B1}–R_{B4}. The coupling between the director boards 190_1–190_8, 210_1–210_8 and the global cache memory 220 is shown in FIG. 8.

More particularly, and referring also to FIG. 2, as noted above, each one of the host computer processors 121_1–121_{32} in the host computer 120 is coupled to a pair of the front-end directors 180_1–180_{32} to provide redundancy in the event of a failure in one of the front end-directors 181_1–181_{32} coupled thereto. Likewise, the bank of disk drives 140 has a plurality of, here 32, disk drives 141_1–141_{32}, each disk drive 141_1–141_{32} being coupled to a pair of the back-end directors 200_1–200_{32} to provide redundancy in the event of a failure in one of the back-end directors 200_1–200_{32} coupled thereto). Thus, considering exemplary host computer processor 121_1, such processor 121_1 is coupled to a pair of front-end directors 180_1, 180_2. Thus, if director 180_1 fails, the host computer processor 121_1 can still access the system interface 160, albeit by the other front-end director 180_2. Thus, directors 180_1 and 180_2 are considered redundancy pairs of directors. Likewise, other redundancy pairs of front-end directors are: front-end directors 180_3, 180_4; 180_5, 180_6; 180_7, 180_8; 180_9, 180_{10}; 180_{11}, 180_{12}; 180_{13}, 180_{14}; 180_{15}, 180_{16}; 180_{17}, 180_{18}; 180_{19}, 180_{20}; 180_{21}, 180_{22}; 180_{23}, 180_{24}; 180_{25}, 180_{26}; 180_{27}, 180_{28}; 180_{29}, 180_{30}; and 180_{31}, 180_{32} (only directors 180_{31} and 180_{32} being shown in FIG. 2).

Likewise, disk drive 141_1 is coupled to a pair of back-end directors 200_1, 200_2. Thus, if director 200_1 fails, the disk drive 141_1 can still access the system interface 160, albeit by the other back-end director 180_2. Thus, directors 200_1 and 200_2 are considered redundancy pairs of directors. Likewise, other redundancy pairs of back-end directors are: back-end directors 200_3, 200_4; 200_5, 200_6; 200_7, 200_8; 200_9, 200_{10}; 200_{11}, 200_{12}; 200_{13}, 200_{14}; 200_{15}, 200_{16}; 200_{17}, 200_{18}; 200_{19}, 200_{20}; 200_{21}, 200_{22}; 200_{23}, 200_{24}; 200_{25}, 200_{26}; 200_{27}, 200_{28}; 200_{29}, 200_{30}; and 200_{31}, 200_{32} (only directors 200_{31} and 200_{32} being shown in FIG. 2). Further, referring also to FIG. 8, the global cache memory 220 includes a plurality of, here eight, cache memory boards 200_1–200_8, as shown. Still further, referring to FIG. 8A, an exemplary one of the cache memory boards, here board 220_1 is shown in detail and is described in detail in U.S. Pat. No. 5,943,287 entitled "Fault Tolerant Memory System", John K. Walton, inventor, issued Aug. 24, 1999 and assigned to the same assignee as the present invention, the entire subject matter therein being incorporated herein by reference. Thus, as shown in FIG. 8A, the board 220_1 includes a plurality of, here four RAM memory arrays, each one of the arrays has a pair of redundant ports, i.e., an A port and a B port. The board itself has sixteen ports; a set of eight A ports M_{A1}–M_{A8} and a set of eight B ports M_{B1}–M_{B8}. Four of the eight A port, here A ports M_{A1}–M_{A4} are coupled to the M_1 port of each of the front-end director boards 190_1, 190_3, 190_5, and 190_7, respectively, as indicated in FIG. 8. Four of the eight B port, here B ports M_{B1}–M_{B4} are coupled to the M_1 port of each of the front-end director boards 190_2, 190_4, 190_6, and 190_8, respectively, as indicated in FIG. 8. The other four of the eight A port, here A ports M_{A5}–M_{A8} are coupled to the M_1 port of each of the back-end director boards 210_1, 210_3, 210_5, and 210_7, respectively, as indicated in FIG. 8. The other four of the eight B port, here B ports M_{B5}-M_{48} are coupled to the M_1 port of each of the back-end director boards 210_2, 210_4, 210_6, and 210_8, respectively, as indicated in FIG. 8

Considering the exemplary four A ports M_{A1}–M_{A4}, each one of the four A ports M_{A1}–M_{A4} can be coupled to the A port of any one of the memory arrays through the logic network 221_{1A}. Thus, considering port M_{A1}, such port can be coupled to the A port of the four memory arrays. Likewise, considering the four A ports M_{A5}–M_{A8}, each one of the four A ports M_{A5}–M_{A8} can be coupled to the A port of any one of the memory arrays through the logic network 221_{1B}. Likewise, considering the four B ports M_{B1}–M_{B4}, each one of the four B ports M_{B1}–M_{B4} can be coupled to the B port of any one of the memory arrays through logic network 221_{1B}. Likewise, considering the four B ports M_{B5}–M_{B8}, each one of the four B ports M_{B5}–M_{B8} can be coupled to the B port of any one of the memory arrays through the logic network 221_{2B}. Thus, considering port M_{B1}, such port can be coupled to the B port of the four memory arrays. Thus, there are two paths data and control from either a front-end director 180_1–180_{32} or a back-end director 200_1–200_{32} can reach each one of the four memory arrays on the memory board. Thus, there are eight sets of redundant ports on a memory board, i.e., ports M_{A1}, M_{B1}; M_{A2}, M_{B2}; M_{A3}, M_{B3}; M_{A4}, M_{B4}; M_{A5}, M_{B5}; M_{A6}, M_{B6}; M_{A7}, M_{B7}; and M_{A8}, M_{B8}. Further, as noted above each one of the directors has a pair of redundant ports, i.e. a 402A port and a 402 B port (FIG. 7). Thus, for each pair of redundant directors, the A port (i.e., port 402A) of one of the directors in the pair is connected to one of the pair of redundant memory ports and the B port (i.e., 402B) of the other one of the directors in such pair is connected to the other one of the pair of redundant memory ports.

More particularly, referring to FIG. 8B, an exemplary pair of redundant directors is shown, here, for example, front-end director 180_1 and front end-director 180_2. It is first noted that the directors 180_1, 180_2 in each redundant pair of directors must be on different director boards, here boards 190_1, 190_2, respectively. Thus, here front-end director boards 190_1–190_8 have thereon: front-end directors 180_1, 180_3, 180_5 and 180_7; front-end directors 180_2, 180_4, 180_6 and 180_8; front end directors 180_9, 180_{11}, 180_{13} and 180_{15}; front end directors 180_{10}, 180_{12}, 180_{14} and 180_{16}; front-end directors 180_{17}, 180_{19}, 180_{21} and 180_{23}; front-end directors 180_{18}, 180_{20}, 180_{22} and 180_{24}; front-end directors 180_{25}, 180_{27}, 180_{29} and 180_{31}; front-end directors 180_{18}, 180_{20}, 180_{22} and 180_{24}. Thus, here back-end director boards 210_1–210_8 have thereon: back-end directors 200_1, 200_3, 200_5 and 200_7; back-end directors 200_2, 200_4, 200_6 and 200_8; back-end directors 200_9, 200_{11}, 200_{13} and 200_{15}; back-end directors 200_{10}, 200_{12}, 200_{14} and 200_{16}; back-end directors 200_{17}, 200_{19}, 200_{21} and 200_{23}; back-end directors 200_{18}, 200_{20}, 200_{22} and 200_{24}; back-end directors 200_{25}, 200_{27}, 200_{29} and 200_{31}; back-end directors 200_{18}, 200_{20}, 200_{22} and 200_{24};

Thus, here front-end director 180_1, shown in FIG. 8A, is on front-end director board 190_1 and its redundant front-end director 180_2, shown in FIG. 8B, is on another front-end director board, here for example, front-end director board 190_2. As described above, the port 402A of the quad port RAM 402 (i.e., the A port referred to above) is connected to switch 406A of crossbar switch 318 and the port 402B of the quad port RAM 402 (i.e., the B port referred to above) is connected to switch 406B of crossbar switch 318. Likewise, for redundant director 180_2, However, the ports M_1–M_4 of switch 406A of director 180_1 are connected to the M_{A1} ports of global cache memory boards 220_1–200_4, as shown, while for its redundancy director 180_2, the ports M_1–M_4 of switch 406A are connected to the redundant M_{B1} ports of global cache memory boards 220_1–200_4, as shown.

Referring in more detail to the crossbar switch 318 (FIG. 7), as noted above, each one of the director boards 190_1–210_8 has such a switch 318 and such switch 318 includes a pair of switches 406A, 406B. Each one of the switches 406A, 406B is identical in construction, an exemplary one thereof, here switch 406A being shown in detail in FIG. 8C. Thus switch 406A includes four input/output director-side ports D_1–D_4 as described in connection with exemplary director board 190_1. Thus, for the director board 190_1 shown in FIG. 7, the four input/output director-side ports D_1–D_4 of switch 406A are each coupled to the port 402A of a corresponding one of the directors 180_1, 180_3, 180_5, and 180_7 on the director board 190_1.

Referring again to FIG. 8C, the exemplary switch 406A includes a plurality of, here four, switch sections 430_1–430_4. Each one of the switch sections 430_1–430_4 is identical in construction and is coupled between a corresponding one of the input/output director-side ports D_1–D_4 and a corresponding one of the output/input memory-side ports M_1–M_4, respectively, as shown. (It should be understood that the output/input memory-side ports of switch 406B (FIG. 7) are designated as ports M_5–M_8, as shown. It should also be understood that while switch 406A is responsive to request signals on busses R_{A1}–R_{A4} from quad port controller 404 in directors 180_1, 180_3, 180_5, 180_7 (FIG. 7), switch 406B is responsive in like manner to request signals on busses R_{B1}–R_{B4} from controller 404 in directors 180_1, 180_3, 180_5 and 180_7). More particularly, controller 404 of director 180_1 produces request signals on busses R_{A1} or R_{B1}. In like manner, controller 404 of director 180_3 produces request signals on busses R_{A2} or R_{B2}, controller 404 of director 180_5 produces request signals on busses R_{A3} or R_{B3}, and controller 404 of direction 180_7 produces request signals on busses R_{A4} or R_{B4}.

Considering exemplary switch section 430_1, such switch section 403_1 is shown in FIG. 8C to include a FIFO 432 fed by the request signal on bus R_{1A}. (It should be understood that the FIFOs, not shown, in switch sections 430_2–430_4 are fed by request signals R_{A2}–R_{A4}, respectively). The switch section 406_1 also includes a request generation 434, and arbiter 436, and selectors 442 and 446, all arranged as shown. The data at the memory-side ports M_1–M_4 are on busses DM1–DM4 are fed as inputs to selector 446. Also fed to selector 446 is a control signal produced by the request generator on bus 449 in response to the request signal R_{A1} stored in FIFO 432. The control signal on bus 449 indicates to the selector 446 the one of the memory-side ports M_1–M_4 which is to be coupled to director-side port D_1. The other switch sections 430_2–430_4 operate in like manner with regard to director-side ports D_1–D_4, respectively and the memory-side ports M_1–M_4.

It is to be noted that the data portion of the word at port D_1 (i.e., the word on bus DD1) is also coupled to the other switch sections 430_2–430_4. It is further noted that the data portion of the words at ports D_2–D_4 (i.e., the words on busses DD2–DD4, respectively), are fed to the switch sections 430_1–430_4, as indicated. That is, each one of the switch sections 430_1–430_4 has the data portion of the words on ports D_1–D_4 (i.e., busses DD1–DD4), as indicated. It is also noted that the data portion of the word at port M_1 (i.e., the word on bus DM1) is also coupled to the other switch sections 430_2–430_4. It if further noted that the data portion of the words at ports M_2–M_4 (i.e., the words on busses DM2–DM4, respectively), are fed to the switch sections 430_2–430_4, as indicated. That is, each one of the switch sections 430_1–430_4 has the data portion of the words on ports M_1–M_4 (i.e., busses DM1–DM4), as indicated.

As will be described in more detail below, a request on bus R_{A1} to switch section 430_1 is a request from the director 180_1 which identifies the one of the four ports M_1–M_4 in switch 430_1 is to be coupled to port 402A of director 180_1 (director side port D_1). Thus, port 402A of director 180_1 may be coupled to one of the memory side ports M_1–M_4 selectively in accordance with the data on bus R_{A1}. Likewise, a request on buses R_{A2}, R_{A3}, R_{A4} to switch section 430_2–430_4, respectively, are requests from the directors 180_3, 180_5, and 180_7, respectively, which identifies the one of the four ports M_1–M_4 in switch 430_1–430_4 is to be coupled to port 402A of directors 180_3, 180_5 and 180_7, respectively.

More particularly, the requests R_{A1} are stored as they are produced by the quad port RAM controller 440 (FIG. 7) in receive FIFO 432. The request generator 434 receives from FIFO 432 the requests and determines which one of the four memory-side ports M_1–M_4 is to be coupled to port 402A of director 180_1. These requests for memory-side ports M_1–M_4 are produced on lines RA1,1–RA1,4, respectively. Thus, in line RA1,1 (i.e., the request for memory side port M_1) is fed to arbiter 436 and the requests from switch sections 430_2–430_4 (which are coupled to port 402A of directors 180_3, 180_5, and 180_7) on line RA2,1, RA3,1 and RA4,1, respectively are also fed to the arbiter 436, as indicated. The arbiter 436 resolves multiple requests for memory-side port M_1 on a first come-first serve basis. The arbiter 436 then produces a control signal on bus 435 indicating the one of the directors 180_1, 180_3, 180_5 or 180_7 which is to be coupled to memory-side port M_1.

The control signal on bus 435 is fed to selector 442. Also fed to selector 442 are the data portion of the data at port D_1, i.e., the data of data bus DD1) along with the data portion of the data at ports D_2–D_4, i.e., the data on data busses DD2–DD4, respectively, as indicated. Thus, the control signal on bus 435 causes the selector 442 to couple to the output thereof the data busses DD1–DD4 from the one of the directors 180_1, 180_3, 180_5, 180_7 being granted access to memory-side port M_1 by the arbiter 436. The selected outputs of selector 442 is coupled to memory-side port M_1. It should be noted that when the arbiter 436 receives a request via the signals on lines RA1,1, RA2,1, RA3,1 and RA4,1, acknowledgements are returned by the arbiter 436 via acknowledgement signals on line AK1,1, Ak1,2, AK1,3, AK1,4, respectively such signals being fed to the request generators 434 in switch section 430_1, 430_2, 430_3, 430_4, respectively.

Thus, the data on any port D_1–D_4 can be coupled to and one of the ports M_1–M_4 to effectuate the point-to-point data paths P_1–P_{64} described above in connection with FIG. 2.

Referring again to FIG. 7, data from host computer 120 (FIG. 2) is presented to the system interface 160 (FIG. 2) in batches from many host computer processors 121_1–121_{32}. Thus, the data from the host computer processors 121_1–121_{32} are interleaved with each other as they are presented to a director 180_1–180_{32}. The batch from each host computer processor 180_1–180_{32} (i.e., source) is tagged by the protocol translator 400. More particularly by a Tacheon ASIC in the case of a fibre channel connection. The controller 404 has a look-up table formed during initialization. As the data comes into the protocol translator 400 and is put into the quad port RAM 420 under the control of controller 404, the protocol translator 400 informs the controller that the data is in the quad port RAM 420. The controller 404 looks at the configuration of its look-up table to determine the global cache memory 220 location (e.g., cache memory board 220_1–220_8) the data is to be stored into. The controller 404 thus produces the request signals on the appropriate bus R_{A1}, R_{B1}, and then tells the quad port RAM 402 that there is a block of data at a particular location in the quad port RAM 402, move it to the particular location in the global cache memory 220. The crossbar switch 318 also takes a look at what other controllers 404 in the directors 180_3, 180_5, and 180_7 on that particular director board 190_1 are asking by making request signal on busses R_{A2}, R_{B2}, R_{A3}, R_{B3}, R_{A4}, R_{B4}, respectively. The arbitration of multiple requests is handled by the arbiter 436 as described above in connection with FIG. 8C.

Referring again to FIG. 7, the exemplary director 180_1 is shown to include in the message engine/CPU controller 314. The message engine/CPU controller 314 is contained in a field programmable gate array (FPGA). The message engine (ME) 315 is coupled to the CPU bus 317 and the DMA section 408 as shown. The message engine (ME) 315 includes a Direct Memory Access (DMA) section 408, a message engine (ME) state machine 410, a transmit buffer 424 and receive buffer 424, a MAC packetizer/depacketizer 428, send and receive pointer registers 420, and a parity generator 321. The DMA section 408 includes a DMA transmitter 418, shown and to be described below in detail in connection with FIG. 9, and a DMA receiver 424, shown and to be described below in detail in connection with FIG. 10, each of which is coupled to the CPU bus interface 317, as shown in FIG. 7. The message engine (ME) 315 includes a transmit data buffer 422 coupled to the DMA transmitter 418, a receive data buffer 424 coupled to the DMA receiver 421, registers 420 coupled to the CPU bus 317 through an address decoder 401, the packetizer/de-packetizer 428, described above, coupled to the transmit data buffer 422, the receive data buffer 424 and the crossbar switch 320, as shown, and a parity generator 321 coupled between the transmit data buffer 422 and the crossbar switch 320. More particularly, the packetizer portion 428P is used to packetize the message payload into a MAC packet (FIG. 2B) passing from the transmit data buffer 422 to the crossbar switch 320 and the de-packetizer portion 428D is used to de-packetize the MAC packet into message payload data passing from the crossbar switch 320 to the receive data buffer 424. The packetization is here performed by a MAC core which builds a MAC packet and appends to each message such things as a source and destination address designation indicating the director sending and receiving the message and a cyclic redundancy check (CRC), as described above. The message engine (ME) 315 also includes: a receive write pointer 450, a receive read pointer 452; a send write pointer 454, and a send read pointer 456.

Referring now to FIGS. 11 and 12, the transmission of a message from a director 180_1–180_{32}, 200_1–200_{32} and the reception of a message by a director 210_1–210_{32}, here exemplary director 180_1 shown in FIG. 7) will be described. Considering first transmission of a message, reference is made to FIGS. 7 and 11. First, as noted above, at power-up the controller 306 (FIG. 5) of both message network boards 304_1, 304_2 initialize the message routing mapping described above for the switches 308_1–308_4 in switch section 308 and for the crossbar switches 320. As noted above, a request is made by the host computer 120. The request is sent to the protocol translator 400. The protocol translator 400 sends the request to the microprocessor 299 via CPU bus 317 and buffer 301. When the CPU 310 (FIG. 7) in the microprocessor 299 of exemplary director 180_1 determines that a message is to be sent to another one of the directors 180_2–180_{32}, 200_1–200_{32}, (e.g., the CPU 310 determines that there has been a "miss" in the global cache memory 220 (FIG. 2) and wants to send a message to the appropriate one of the back-end directors 200_1–200_{32}, as described above in connection with FIG. 2), the CPU 310 builds a 64 byte descriptor (FIG. 2A) which includes a 32 byte message payload indicating the addresses of the batch of data to be read from the bank of disk drives 140 (FIG. 2) (Step 500) and a 32 byte command field (Step 510) which indicates the message destination via an 8-byte bit vector, i.e., the director, or directors, which are to receive the message. An 8-byte portion of the command field indicates the director or directors, which are to receive the message. That is, each one of the 64 bits in the 8-byte portion corresponds to one of the 64 directors. Here, a logic 1 in a bit indicates that the corresponding director is to receive a message and a logic 0 indicates that such corresponding director is not to receive the message. Thus, if the 8-byte word has more than one logic 1 bit more than one director will receive the same message. As will be described, the same message will not be sent in parallel to all such directors but rather the same message will be sent sequentially to all such directors. In any event, the resulting 64-byte descriptor is generated by the CPU 310 (FIG. 7) (Step 512) is written into the RAM 312 (Step 514), as shown in FIG. 11.

More particularly, the RAM 512 includes a pair of queues; a send queue and a receive queue, as shown in FIG. 7. The RAM 312 is coupled to the CPU bus 317 through an Error Detection and Correction (EDAC)/Memory control section 303, as shown. The CPU 310 then indicates to the message engine (ME) 315 state machine 410 (FIG. 7) that a descriptor has been written into the RAM 312. It should be noted that the message engine (ME) 315 also includes: a receive write pointer or counter 450, the receive read pointer or counter 452, the send write pointer or counter 454, and the send read pointer or counter 454, shown in FIG. 7. All four pointers 450, 452, 454 and 456 are reset to zero on power-up. As is also noted above, the message engine/CPU controller 314 also includes: the de-packetizer portion 428D of packetizer/de-packetizer 428, coupled to the receive data buffer 424 (FIG. 7) and a packetizer portion 428P of the packetizer/de-packetizer 428, coupled to the transmit data buffer 422 (FIG. 7). Thus, referring again to FIG. 11, when the CPU 310 indicates that a descriptor has been written into the RAM 312 and is now ready to be sent, the CPU 310 increments the send write pointer and sends it to the send

write pointer register 454 via the register decoder 401. Thus, the contents of the send write pointer register 454 indicates the number of messages in the send queue 312S of RAM 312, which have not been sent. The state machine 410 checks the send write pointer register 454 and the send read pointer register 456, Step 518. As noted above, both the send write pointer register 454 and the send read pointer register 456 are initially reset to zero during power-up. Thus, if the send read pointer register 456 and the send write pointer register 454 are different, the state machine knows that there is a message is in RAM 312 and that such message is ready for transmission. If a message is to be sent, the state machine 410 initiates a transfer of the stored 64-byte descriptor to the message engine (ME) 315 via the DMA transmitter 418, FIG. 7 (Steps 520, 522). The descriptor is sent from the send queues 312S in RAM 312 until the send read pointer 456 is equal to the send write pointer 454.

As described above in connection with Step 510, the CPU 310 generates a destination vector indicating the director, or directors, which are to receive the message. As also indicated above the command field is 32-bytes, eight bytes thereof having a bit representing a corresponding one of the 64 directors to receive the message. For example, referring to FIG. 11C, each of the bit positions 1–64 represents directors 180_1–180_{32}, 200_1–200_{31}, respectively. Here, in this example, because a logic 1 is only in bit position 1, the eight-byte vector indicates that the destination director is only front-end director 108_1. In the example in FIG. 11D, because a logic 1 is only in bit position 2, the eight-byte vector indicates that the destination director is only front-end director 108_2. In the example in FIG. 11E, because a logic 1 is more than one bit position, the destination for the message is to more than one director, i.e., a multi-cast message. In the example in FIG. 11E, a logic 1 is only in bit positions 2, 3, 63 and 64. Thus, the eight-byte vector indicates that the destination directors are only front-end director 108_2 and 108_3 and back-end directors 200_{31} and 200_{32}. There is a mask vector stored in a register of register section 420 (FIG. 7) in the message engine (ME) 315 which identifies director or directors which may be not available to use (e.g. a defective director or a director not in the system at that time), Step 524, 525, for a uni-cast transmission). If the message engine (ME) 315 state machine 410 indicates that the director is available by examining the transmit vector mask (FIG. 11F) stored in register 420, the message engine (ME) 315 encapsulates the message payload with a MAC header and CRC inside the packetizer portion 428P, discussed above (Step 526). An example of the mask is shown in FIG. 11F. The mask has 64 bit positions, one for each one of the directors. Thus, as with the destination vectors described above in connection with FIGS. 11C–11E, bit positions 1–64 represents directors 180_1–180_{32}, 200_1–200_{32}, respectively. Here in this example, a logic 1 in a bit position in the mask indicates that the representative director is available and a logic 0 in such bit position indicates that the representative director is not available. Here, in the example shown in FIG. 11F, only director 200_{32} is unavailable. Thus, if the message has a destination vector as indicated in FIG. 11E, the destination vector, after passing through the mask of FIG. 11F modifies the destination vector to that shown in FIG. 11G. Thus, director 200_{32} will not receive the message. Such mask modification to the destination vector is important because, as will be described, the messages on a multi-cast are sent sequentially and not in parallel. Thus, elimination of message transmission to an unavailable director or directors increases the message transmission efficiency of the system.

Having packetized the message into a MAC packet via the packetizer portion of the packetizer/de-packetizer 428 (FIG. 7), the message engine (ME) 315 transfers the MAC packet to the crossbar switch 320 (Step 528) and the MAC packet is routed to the destination by the message network 260 (Step 530) via message network boards 304_1, 304_2 or on the same director board via the crossbar switch 320 on such director board.

Referring to FIG. 12, the message read operation is described. Thus, in Step 600 the director waits for a message. When a message is received, the message engine (ME) 315 state machine 410 receives the packet (Step 602). The state machine 410 checks the receive bit vector mask (FIG. 11 stored in register 426) against the source address of the packet (Step 604). If the state machine 410 determines that the message is from an improper source (i.e., a faulty director as indicated in the mask, FIG. 11F, for example), the packet is discarded (Step 606). On the other hand, if the state machine 410 determines that the packet is from a proper or valid director (i.e., source), the message engine (ME) 315 de-encapsulates the message from the packet (Step 608) in de-packetizer 428D. The state machine 410 in the message engine (ME) 315 initiates a 32-byte payload transfer via the DMA receive operation (Step 610). The DMA writes the 32 byte message to the memory receive queue 313R in the RAM 312 (Step 612). The message engine (ME) 315 state machine 410 then increments the receive write pointer register 450 (Step 614). The CPU 310 then checks whether the receive write pointer 50 is equal to the receive read pointer 452 (Step 616). If they are equal, such condition indicates to the CPU 310 that a message has not been received (Step 618). On the other hand, if the receive write pointer 450 and the receive read pointer 452 are not equal, such condition indicates to the CPU 310 that a message has been received and the CPU 310 processes the message in the receive queue 314R of RAM 312 and then the CPU 310 increments the receive read pointer and writes it into the receive read pointer register 452. Thus, messages are stored in the receive queue 312R of RAM 312 until the contents of the receive read pointer 452 and the contents of the receive write pointer 450, which are initialized to zero during power-up, are equal.

Referring now to FIG. 13, the acknowledgement of a message operation is described. In Step 700 the receive DMA engine 420 successfully completes a message transfer to the receive queue in RAM 312 (FIG. 7). The state machine 410 in the message engine (ME) 315 generates an acknowledgement MAC packet and transmits the MAC packet to the sending director via the message network 260 (FIG. 2) (Steps 702, 704). The message engine (ME) 315 at the sending director de-encapsulates a 16 byte status payload in the acknowledgement MAC packet and transfers such status payload via a receive DMA operation (Step 706). The DMA of the sending (i.e., source) director writes to a status field of the descriptor within the RAM memory send queue 314S (Step 708).The state machine 410 of the message engine (ME) 315 of the sending director (which received the acknowledgement message) increments its send read pointer 454 (Step 712). The CPU 310 of the sending director (which received the acknowledgement message) processes the descriptor status and removes the descriptor from the send queue 312S of RAM 312 (Step 714). It should be noted that the send and receive queues 312S and 312R are each circular queues

As noted above, the MAC packets are, during normal operation, transmitted alternatively to one of the pair of message network boards 304_1, 304_2 by hardware a selector S in the crossbar switch 320. The selector S is responsive to the bit B in the header of the MAC packet (FIG. 2B) and, when such bit B is one logic state the data is coupled to one of the message networks boards 402A and in response to the opposite logic state the data is coupled to the other one of the message networks boards 402B. That is, when one message is transmitted to board 304_1 the next message is transmitted to board 304_2.

Referring again to FIG. 9, the details of an exemplary transmit DMA 418 is shown. As noted above, after a decriptor has been created by the CPU 310 (FIG. 7) and is then stored in the RAM 312. If the send write pointer 450 (FIG. 7) and send read pointer 452, described above, have different counts an indication is provided by the state machine 410 in the message engine (ME) 315 (FIG. 7) that the created descriptor is available for DMA transmission to the message engine (ME) 315, the payload off the descriptor is packetized into a MAC packet and sent through the message network 360 (FIG. 2) to one or more directors 180_1–180_{32}, 200_1–200_{32}. More particularly, the descriptor created by the CPU 310 is first stored in the local cache memory 319 and is later transferred to the send queue 312S in RAM 312. When the send write pointer 450 and send read pointer 452 have different counts, the message engine (ME) 315 state machine 410 initiates a DMA transmission as discussed above in connection with Step 520 (FIG. 11). Further, as noted above, the descriptor resides in send queues 312R within the RAM 312. Further, as noted above, each descriptor which contains the message is a fixed size, here 64-bytes. As each new, non-transmitted descriptor is created by the CPU 310, it is sequentially stored in a sequential location, or address in the send queue 312S. Here, the address is a 32-bit address.

When the transmit DMA is initiated, the state machine 410 in the message engine (ME) 315 (FIG. 7), sends the queue address on bus 411 to an address register 413 in the DMA transmitter 418 (FIG. 9) along with a transmit write enable signal Tx_WE signal. The DMA transmitter 418 requests the CPU bus 317 by asserting a signal on Xmit_Br. The CPU bus arbiter 414 (FIG. 7) performs a bus arbitration and when appropriate the arbiter 414 grants the DMA transmitter 418 access to the CPU bus 317. The Xmit Cpu state machine 419 then places the address currently available in the address register 413 on the Address bus portion 317A of CPU bus 317 by loading the output address register 403. Odd parity is generated by a Parity generator 405 before loading the output address register 403. The address in register 403 is placed on the CPU bus 317 (FIG. 7) for RAM 312 send queue 312S, along with appropriate read control signals via CPU bus 317 portion 317C. The data at the address from the RAM 312 passes, via the data bus portion 317D of CPU bus 317, through a parity checker 415 to a data input register 417. The control signals from the CPU 310 are fed to a Xmit CPU state machine 419 via CPU bus 317 bus portion 317C. One of the control signals indicates whether the most recent copy of the requested descriptor is in the send queue 312S of the RAM 312 or still resident in the local cache memory 319. That is, the most recent descriptor at any given address is first formed by the CPU 310 in the local cache memory 319 and is later transferred by the CPU 310 to the queue in the RAM 312. Thus, there may be two descriptors with the same address; one in the RAM 312 and one in the local cache memory 319 (FIG. 7), the most recent one being in the local cache memory 319. In either event, the transmit DMA 418 must obtain the descriptor for DMA transmission from the RAM 312 and this descriptor is stored in the transmit buffer register 421 using signal 402 produced by the state machine 419 to load these registers 421. The control signal from the CPU 310 to the Xmit CPU state machine 419 indicates whether the most recent descriptor is in the local cache memory 319. If the most recent descriptor is in the local cache memory 319, the Xmit CPU state machine 419 inhibits the data that was just read from send queue 312S in the RAM 312 and which has been stored in register 421 from passing to selector 423. In such case, state machine 419 must perform another data transfer at the same address location. The most recent message is then transferred by the CPU 310 from the local cache memory 319 to the send queue 312S in the RAM 312. The transmit message state machine 419 then re-arbitrates for the CPU bus 317 and after it is granted such CPU bus 317, the Xmit CPU state machine 419 then reads the descriptor from the RAM 312. This time, however, there the most recent descriptor is available in the send queue 312*s* in the RAM 312. The descriptor in the RAM 312 is now loaded into the transmit buffer register 421 in response to the assertion of the signal 402 by the Xmit CPU state machine 419. The descriptor in the register 421 is then transferred through selector 423 to message bus interface 409 under the control of a Xmit message (msg) state machine 427. That is, the descriptor in the transmit buffer register 421 is transferred to the transmit data buffer 422 (FIG. 7) over the 32 bit transmit message bus interface 409 by the Xmit message (msg) state machine 427. The data in the transmit data buffer 422 (FIG. 7) is packetized by the packetizer section of the packetizer/de-packetizer 428 as described in Step 530 in FIG. 11.

More particularly, and referring also to FIG. 14A, the method of operating the transmit DMA 418 (DIG. 9) is shown. As noted above, each descriptor is 64-byte. Here, the transfer of the descriptor takes place over two interfaces namely, the CPU bus 317 and the transmit message interface bus 409 (FIG. 7). The CPU bus 317 is 64 bits wide and eight, 64-bit double-words constitute a 64-byte descriptor. The Xmit CPU state machine 419 generates the control signals which result in the transfer of the descriptor from the RAM 312 into the transmit buffer register 421 (FIG. 7). The 64-byte descriptor is transferred in two 32-byte burst accesses on the CPU bus 317. Each one of the eight double words is stored sequentially in the transmit buffer register 421 (FIG. 9). Thus, in Step 800, the message engine 315 state machine 410 loads the transmit DMA address register 413 with the address of the descriptor to be transmitted in the send queue 312S in RAM 312. This is done by the asserting the Tx_WE signal and thus puts Xmit CPU state machine 419 in step 800, loads the address register 413 and proceeds to step 802. In step 802, the Xmit Cpu state machine 419 loads the CPU transfer counter 431 (FIG. 9) with a 32-byte count, which is 2. This is the number of 32 byte transfers that would be required to transfer the 64-byte descriptor, Step 802. The Xmit Cpu state machine 419 now proceeds to Step 804. In step 804, the transmit DMA state machine 419 checks the validity of the address that is loaded into its address register 413. The address loaded into the address register 413 is checked against the values loaded into the memory address registers 435. The memory address registers 435 contain the base address and the offset of the send queue 312*s* in the RAM 312. The sum of the base address and the offset is the range of addresses for the send queue 312S in RAM 312. The address check circuitry 437 constantly checks whether the address in the address register 413 is with in the range of the send queue 312S in the RAM 312. If the address is found to be outside the range of the send queue 312S the transfer is aborted, this status is stored in the status register 404 and then passed back to the message engine 315 state machine 410 in Step 416. The check for valid addresses is done in Step 805. If the address is within the range, i.e., valid, the transmit DMA state machine 419 proceeds with the transfer and proceeds to Step 806. In the step 806, the transmit DMA state machine 419 requests the CPU bus 317 by asserting the Xmit_BR signal to the arbiter 414 and then proceeds to Step 807. In Step 807, the Xmit Cpu state machine 419 constantly checks if it has been granted the bus by the arbiter. When the CPU bus 317 is granted, the Xmit CPU state machine proceeds to Step 808. In Step 808, the Xmit Cpu state machine 419 generates an address and a data cycle which essentially reads 32-bytes of the descriptor from the send queue 312S in the RAM 312 into its transmit buffer register 421. The Xmit Cpu state machine 419 now proceeds to step 810. In Step 810, the Xmit Cpu state machine 419 loads the descriptor that was read into its buffer registers 421 and proceeds to Step 811. In Step 811, a check is made for any local cache memory 319 coherency errors (i.e., checks whether the most recent data is in the cache memory 319 and not in the RAM 312) on these 32-bytes of data. If this data is detected to be resident in the local CPU cache memory 319, then the Xmit Cpu state machine 419 discards this data and proceeds to Step 806. The Xmit Cpu state machine 419 now requests for the CPU bus 317 again and when granted, transfers another 32-bytes of data into the transmit buffer register 421, by which time the CPU has already transferred the latest copy of the descriptor into the RAM 312. In cases when the 32-bytes of the descriptor initially fetched from the RAM 312 was not resident in the local CPU cache memory 319 (i.e., if no cache coherency errors were detected), the Xmit Cpu state machine 419 proceeds to Step 812. In Step 812, the Xmit CPU state machine 419 decrements counters 431 and increments the address register 413 so that such address register 413 points to the next address. The Xmit Cpu state machine then proceeds to step 814. When in Step 814, the Transmit CPU state machine 419 checks to see if the transfer counter 431 has expired, i.e., counted to zero, if the count was found to be non-zero, it then, proceeds to Step 804 to start the transfer of the next 32-bytes of the descriptor. In case the counter 431 is zero, the process goes to Step 816 to complete the transfer. The successful transfer of the second 32-bytes of descriptor from the RAM 312 into the transmit DMA buffer register 421 completes the transfer over the CPU bus 317.

The message interface 409 is 32 bits wide and sixteen, 32 bit words constitute a 64-byte descriptor. The 64-byte descriptor is transferred in batches of 32 bytes each. The Xmit msg state machine 427 controls and manages the interface 409. The Xmit Cpu state machine asserts the signal 433 to indicate that the first 32 bytes have been successfully transferred over the CPU bus 317 (Step 818, FIG. 14B), this puts the Xmit msg state machine into Step 818 and starts the transfer on the message interface. In step 820, the Xmit msg machine 427 resets burst/transfer counters 439 and initiates the transfer over the message interface 409. In Step 820, the transfer is initiated over the message interface 409 by asserting the "transfer valid" (TX_DATA_Valid) signal indicating to the message engine 315 state machine 410 that valid data is available on the bus 409. The transmit msg machine 427 transfers 32 bits of data on every subsequent clock until its burst counter in burst/transfer counter 439 reaches a value equal to eight, Step 822. The burst counter in burst/transfer counter 439 is incremented with each 32-bit word put on the message bus 409 by a signal on line 433. When the burst count is eight, a check is made by the state machine 427 as to whether the transmit counter 431 has expired, i.e., is zero, Step 824. The expiry of the transfer counter in burst/transfer counter 439 indicates the 64 byte descriptor has been transferred to the transmit buffer 422 in message engine 315. If it has expired, the transmit message state machine 427 proceeds to Step 826. In step 826, the Xmit msg state machine asserts the output End of Transfer (Tx_EOT) indicating the end of transfer over the message bus 409. In this state, after the assertion of the Tx_EOT signal the status of the transfer captured in the status register 404 is sent to the message engine 315 state machine 410. The DMA operation is complete with the descriptor being stored in the transmit buffer 422 (FIG. 7).

On the other hand, if the transfer counter in burst/transfer counter 439 has not expired, the process goes to Step 800 and repeats the above described procedure to transfer the 2nd 32 bytes of descriptor data, at which time the transfer will be complete.

Referring now to FIG. 10, the receiver DMA 420 is shown. Here, a message received from another director is to be written into the RAM 312 (FIG. 7). The receive DMA 420 is adapted to handle three types of information: error information which is 8 bytes in size; acknowledgement information which is 16 bytes in size; and receive message payload and/or fabric management information which is 32 byes in size. Referring also to FIG. 7, the message engine 315 state machine 410 asserts the Rx_WE signal, indicating to the Receive DMA 420 that is ready transfer the Data in its Rec buffer 416 FIG. 7. The data in the Receive buffer could be the 8-byte error information, the 16-byte Acknowledgement information or the 32-byte Fabric management/Receive message payload information. It places a 2 bit encoded receive transfer count, on the Rx_transfer count signal indicating the type of information and an address which is the address where this information is to be stored in the receive queue of RAM 312. In response to the receive write enable signal Rx_WE, the Receive message machine 450 (FIG. 10) loads the address into the address register 452 and the transfer count indicating the type of information, into the receive transfer counter 454. The address loaded into the address register 452 is checked by the address check circuitry 456 to see if it is with in the range of the Receive queue addresses, in the RAM 312. This is done by checking the address against the values loaded into the memory registers 457 (i.e., a base address register and an offset register therein). The base address register contains the start address of the receive queue 312R residing in the RAM 312 and the offset register contains the size of this receive queue 312R in RAM 312. Therefore the additive sum of, the values stored in the base address register and the offset register specifies the range of addresses of the receive queue in the RAM 312R. The memory registers 457 are loaded during initialization. On the subsequent clock after the assertion of the Rx_WE signal, the message engine 315 state machine 410 the proceeds to place the data on a 32-bit message engine 315 data bus 407, FIG. 10. A Rx_data_valid signal accompanies each 32 bits of data, indicating that the data on the message engine data bus 407 is valid. In response to this Rx_data_valid signal the receive message state machine 450 loads the data on the data bus into the receive buffer register 460. The end of the transfer over the message engine data bus 407*d* is indicated by the assertion of the Rx_EOT signal at which time the Receive message state machine 450 loads the last 32 bits of data on the message engine data bus 407D of bus 407, into the receive buffer registers 460. This signals the end of the transfer over the message engine data bus 407D portion of bus 407. At the end of such transfer is conveyed to the Rx_Cpu state machine 462 by the assertion of the signal 464. The Receive CPU machine 462 now, requests for the CPU bus 317 by asserting the signal REC_Br. After an arbitration by CPU bus arbiter 414 (FIG. 7) the receive DMA 420 (FIG. 10) is given access to the CPU bus 317. The Receive CPU state machine 462 proceeds to transfer the data in its buffer registers 424 over the CPU bus 317 into the Receive queue 312R in the RAM 312. Simultaneously, this data is also transferred into a duplicate buffer register 466. The data at the output of the receive buffer register 460 passes to one input of a selector 470 and also passes to a duplicate data receive buffer register 460. The output of the duplicate receive buffer register 466 is fed to a second input of the selector 470. As the data is being transferred by the Receive CPU state machine 462, it is also checked for cache coherency errors. If the data corresponding to the address being written into the RAM 312, is located in the CPU's local cache memory 319 (FIG. 7), the receive DMA machine 420 waits for the CPU 310 to copy the old data in its local cache memory 319 back to the receive queue 312R in the RAM 312 and then overwrites this old data with a copy of the new data from the duplicate buffer register 466.

More particularly, if central processing unit 310 indicates to the DMA receiver 420 that the data the receive buffer register 460 is available in the local cache memory 319, the receive CPU state machine 462 produces a select signal on line 463 which couples the data in the duplicate buffer register 466 to the output of selector 470 and then to the bus 317 for store in the random access memory 312.

The successful write into the RAM 312 completes the DMA transfer. The receive DMA 420 then signals the message engine 315 state machine 410 on the status of the transfer. The status of the transfer is captured in the status register 459.

Thus, with both the receive DMA and the transmit DMA, there is a checking of the local cache memory 319 to determine whether it has "old" data, in the case of the receive DMA or whether it has "new data" in the case of the transmit DMA.

Referring now to FIG. 15A, the operation of the receive DMA 420 is shown. Thus, in Step 830 the Receive message machine 450 checks if the write enable signal Rx_WE is asserted. If found asserted, the receive DMA 420 proceeds to load the address register 452 and the transfer counter 454. The value loaded into the transfer counter 454 determines the type of DMA transfer requested by the Message engine state machine 310 in FIG. 7. The assertion of the Rx_DATA_VALID signal is asserted. If asserted it proceeds to step 836. The Rx msg state machine loads the buffer register 460 (FIG. 19) in Step 836 with the data on the message engine data bus 407D of bus 407 FIG. 10. The Rx_DATA_VALID signal accompanies each piece of data put on the bus 407. The data is sequentially loaded into the buffer registers 460 (FIG. 10). The End of the transfer on the message engine data bus 407D of bus 407 is indicated by the assertion of the Rx_EOT signal. When the Receive message state machine 450 is in the End of transfer state Step 840 it signals the Receive CPU state machine 462 and this starts the transfer on the CPU bus 317 side.

The flow for the Receive CPU state machine is explained below. Thus, referring to FIG. 15B, the End of the transfer on the Message engine data bus 407D portion of bus 407 starts the Receive CPU state machine 462 and puts it in Step 842. The Receive CPU state machine 462 checks for validity of the address in this state (Step 844). This is done by the address check circuitry 456. If the address loaded in the address register 452 is outside the range of the receive queue 312R in the RAM 312, the transfer is aborted and the status is captured in the Receive status register 459 and the Rec Cpu state machine 462 proceeds to Step 845. On a valid address the Receive CPU state machine 462 goes to Step 846. In Step 846 the Receive Cpu state machine 462 requests for access of the CPU bus 31. It then proceeds to Step 848. In step 848 it checks for a grant on the bus 317. On a qualified grant it proceeds to Step 850. In Step 850, the Rec Cpu state machine 462 performs an address and a data cycle, which essentially writes the data in the buffer registers 460 into the receive queue 312R in RAM 312. Simultaneously with the write to the RAM 312, the data put on the CPU bus 317 is also loaded into the duplicate buffer register 466. At same time, the CPU 310 also indicates on one of the control lines, if the data corresponding to the address written to in the RAM 312 is available in its local cache memory 319. At the end of the address and data cycle the Rec Cpu state machine 462 proceeds to Step 850. In this step it checks for cache coherency errors of the type described above in connection with the transmit DMA 418 (FIG. 9). If cache coherency error is detected and the receive CPU state machine 462 proceeds to Step 846 and retries the transaction more particularly, the Receive CPU state machine 462 now generates another address and data cycle to the previous address and this time the data from the duplicate buffer 466 is put on to the CPU data bus 317. If there were no cache coherency errors the Receive CPU state machine 462 proceeds to Step 852 where it decrements the transfer counter 454 and increment the address in the address register 452. The Receive Cpu state machine 462 then proceeds to Step 854. In Step 854, the state machine 462 checks if the transfer counter has expired, i.e., is zero. On a non zero transfer count the receive Cpu state machine 462 proceeds to Step 844 and repeats the above described procedure until the transfer becomes zero. A zero transfer count when in step 854 completes the write into the receive queue 312R in RAM 312 and the Rec Cpu state machine proceeds to 845. In step 845, it conveys status stored in the status register back to status is conveyed to the message engine 315 state machine 410.

Referring again to FIG. 7, the interrupt control status register 412 will be described in more detail. As described above, a packet is sent by the pocketsize portion of the packetizer/de-packetizer 428 to the crossbar switch 320 for transmission to one or more of the directors. It is to be noted that the packet sent by the packetizer portion of the packetizer/de-packetizer 428 passes through a parity generator PG in the message engine 315 prior to passing to the crossbar switch 320. When such packet is sent by the message engine 315 in exemplary director 180_1, to the crossbar switch 320, a parity bit is added to the packet by parity bit generator PG prior to passing to the crossbar switch 320. The parity of the packet is checked in the parity checker portion of a parity checker/generator (PG/C) in the crossbar switch 320. The result of the check is sent by the PG/C in the crossbar switch 320 to the interrupt control status register 412 in the director 180_1.

Likewise, when a packet is transmitted from the crossbar switch 320 to the message engine 315 of exemplary director 180_1, the packet passes through a parity generator portion of the parity checker/generator (PG/C) in the crossbar switch 320 prior to being transmitted to the message engine 315 in director 180_1. The parity of the packet is then checked in the parity checker portion of the parity checker (PC) in direction 180_1 and is the result (i.e., status) is transmitted to the status register 412.

A number of embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Accordingly, other embodiments are within the scope of the following claims.

What is claimed is:

1. A system interface comprising:
a plurality of first directors;
a plurality of second directors;
a data transfer section having a cache memory, such cache memory being coupled to the plurality of first and second directors;
a messaging network, operative independently of the data transfer section, coupled to the plurality of first directors and the plurality of second directors; and
wherein the first and second directors control data transfer between the first directors and the second directors in response to messages passing between the first directors and the second directors through the messaging network to facilitate data transfer between first directors and the second directors with such data passing through the cache memory in the data transfer section;
wherein each one of the first directors includes:
a data pipe coupled between an input of such one of the first directors and the cache memory;
a microprocessor;
a controller;
a common bus, such bus interconnecting the data pipe, the microprocessor, and the controller; and wherein
the controller controls the transfer of the messages between the message network and such one of the first directors and the data between the input of such one of the first directors and the cache memory.

2. The system interface recited in claim 1 wherein each one of the second directors includes:
a data pipe coupled between an input of such one of the second directors and the cache memory;
a microprocessor;
a controller;
a common bus, such bus interconnecting the data pipe, the microprocessor, and the controller; and wherein
the controller controls the transfer of the messages between the message network and such one of the second directors and the data between the input of such one of the second directors and the cache memory.

3. The system interface recited in claim 1 wherein each one of the controller includes a bus arbiter coupled to the common bus for arbitrating access to such common bus.

4. The system recited in claim 2 wherein each one of the controllers in the second directors includes a bus arbiter coupled to the common bus for arbitrating access to such common bus.

5. A data storage system for transferring data between a host computer/server and a bank of disks drives through a system interface, such system interface comprising:
a plurality of first directors coupled to host computer/server;
a plurality of second directors coupled to the bank of disk drives;
a data transfer section having a cache memory, such cache memory being coupled to the plurality of first and second directors;
a messaging network, operative independently of the data transfer section, coupled to the plurality of first directors and the plurality of second directors; and
wherein the first and second directors control data transfer between the host computer and the bank of

disk drives in response to messages passing between the first directors and the second directors through the messaging network to facilitate the data transfer between host computer/server and the bank of disk drives with such data passing through the cache memory in the data transfer section;

wherein each one of the first directors includes:

a data pipe coupled between an input of such one of the first directors and the cache memory;

a microprocessor;

a controller;

a common bus, such bus interconnecting the data pipe, the microprocessor, and the controller; and wherein

the controller controls the transfer of the messages between the message network and such one of the first directors and the data between the input of such one of the first directors and the cache memory.

6. The storage system recited in claim 5 wherein each one of the second directors includes:

a data pipe coupled between an input of such one of the second directors and the cache memory;

a microprocessor;

a controller;

a common bus, such bus interconnecting the data pipe, the microprocessor, and the controller; and wherein

the controller controls the transfer of the messages between the message network and such one of the second directors and the data between the input of such one of the second directors and the cache memory.

7. The storage system recited in claim 5 wherein each one of the controllers includes a bus arbiter coupled to the common bus for arbitrating access to such common bus.

8. The storage system recited in claim 6 wherein each one of the controller in the second directors includes a bus arbiter coupled to the common bus for arbitrating access to such common bus.

9. A method of operating a system interface having a plurality of first directors, a plurality of second directors and a data transfer section having a cache memory, such cache memory being coupled to the plurality of first and second directors, such method comprising:

providing a messaging network, operative independently of the data transfer section, coupled to the plurality of first directors and the plurality of second directors to control data transfer between the first directors and the second directors in response to messages passing between the first directors and the second directors through the messaging network to facilitate data transfer between first directors and the second directors with such data passing through the cache memory in the data transfer section; and providing each one of the first directors with:

a data pipe coupled between an input of such one of the first directors and the cache memory;

a microprocessor;

a controller;

a common bus, such bus interconnecting the data pipe, the microprocessor, and the controller; and wherein

the controller controls the transfer of the messages between the message network and such one of the first directors and the data between the input of such one of the first directors and the cache memory.

10. The method recited in claim 9 including providing each one of the second directors with:

a data pipe coupled between an input of such one of the second directors and the cache memory;

a microprocessor;

a controller;

a common bus, such bus interconnecting the data pipe, the microprocessor, and the controller; and wherein

the controller controls the transfer of the messages between the message network and such one of the second directors and the data between the input of such one of the second directors and the cache memory.

11. The method recited in claim 9 including providing each one of the controllers with a bus arbiter coupled to the common bus for arbitrating access to such common bus.

12. The method recited in claim 10 including providing each one of the controller in the second directors with a bus arbiter coupled to the common bus for arbitrating access to such common bus.

* * * * *

US005274645A

United States Patent [19]

Idleman et al.

[11] Patent Number: 5,274,645

[45] Date of Patent: * Dec. 28, 1993

[54] DISK ARRAY SYSTEM

[75] Inventors: Thomas E. Idleman, Santa Clara; Robert S. Koontz, Atherton; David T. Powers, Morgan Hill; David H. Jaffe, Belmont; Larry P. Henson, Santa Clara; Joseph S. Glider, Palo Alto; Kumar Gajjar, San Jose, all of Calif.

[73] Assignee: Micro Technology, Inc., Sunnyvale, Calif.

[*] Notice: The portion of the term of this patent subsequent to Aug. 18, 2009 has been disclaimed.

[21] Appl. No.: 872,560

[22] Filed: Apr. 23, 1992

Related U.S. Application Data

[63] Continuation of Ser. No. 601,482, Oct. 22, 1990, which is a continuation-in-part of Ser. No. 505,622, Apr. 6, 1990, and a continuation-in-part of Ser. No. 506,703, Apr. 6, 1990, and a continuation-in-part of Ser. No. 488,749, Mar. 2, 1990.

[51] Int. Cl.^{5} G06F 11/20

[52] U.S. Cl. 371/10.1; 371/11.1; 371/40.1; 395/575

[58] Field of Search 371/10.1, 10.2, 11.1, 371/40.1; 395/575

[56] References Cited

U.S. PATENT DOCUMENTS

4,817,035	3/1989	Timsit	371/10.1
4,914,656	4/1990	Dunphy, Jr. et al.	371/10.2
4,958,351	9/1990	Flora et al.	371/40.1
4,989,205	1/1991	Dunphy, Jr. et al.	371/10.1
5,088,081	2/1992	Farr	371/10.1

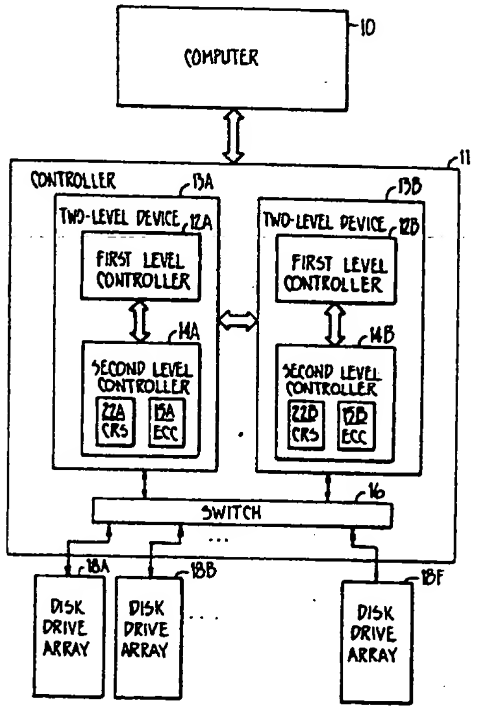
Primary Examiner—Charles E. Atkinson

Attorney, Agent, or Firm—Townsend and Townsend Khourie and Crew

[57] ABSTRACT

A method and apparatus for controlling data flow between a computer and a group of memory devices arranged in a particular logical configuration. The system includes a group of first level controllers and a group of second level controllers. The first level controllers and the second level controllers work together such that if one of the second level controllers fails, the routing between the first level controllers and the memory devices is switched to a properly functioning second level controller without the need to involve the computer in the rerouting process. The logical configuration of the memory devices remains constant. The invention also includes switching circuitry which permits a functioning second level controller to assume control of a group of memory devices formerly primarily controlled by the failed second level controller. In addition, the invention provides error check and correction as well as mass storage device configuration circuitry.

29 Claims, 31 Drawing Sheets



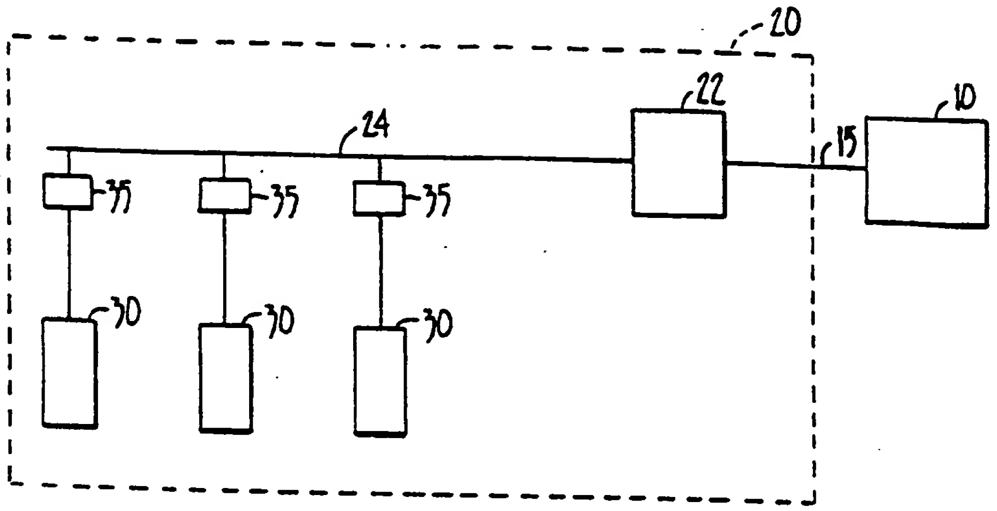


FIG_1. (PRIOR ART)

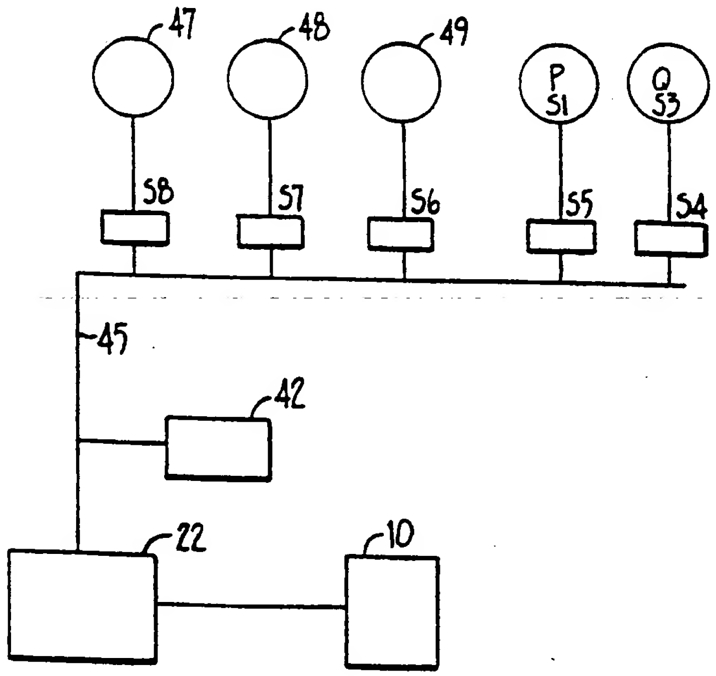


FIG_2. (PRIOR ART)

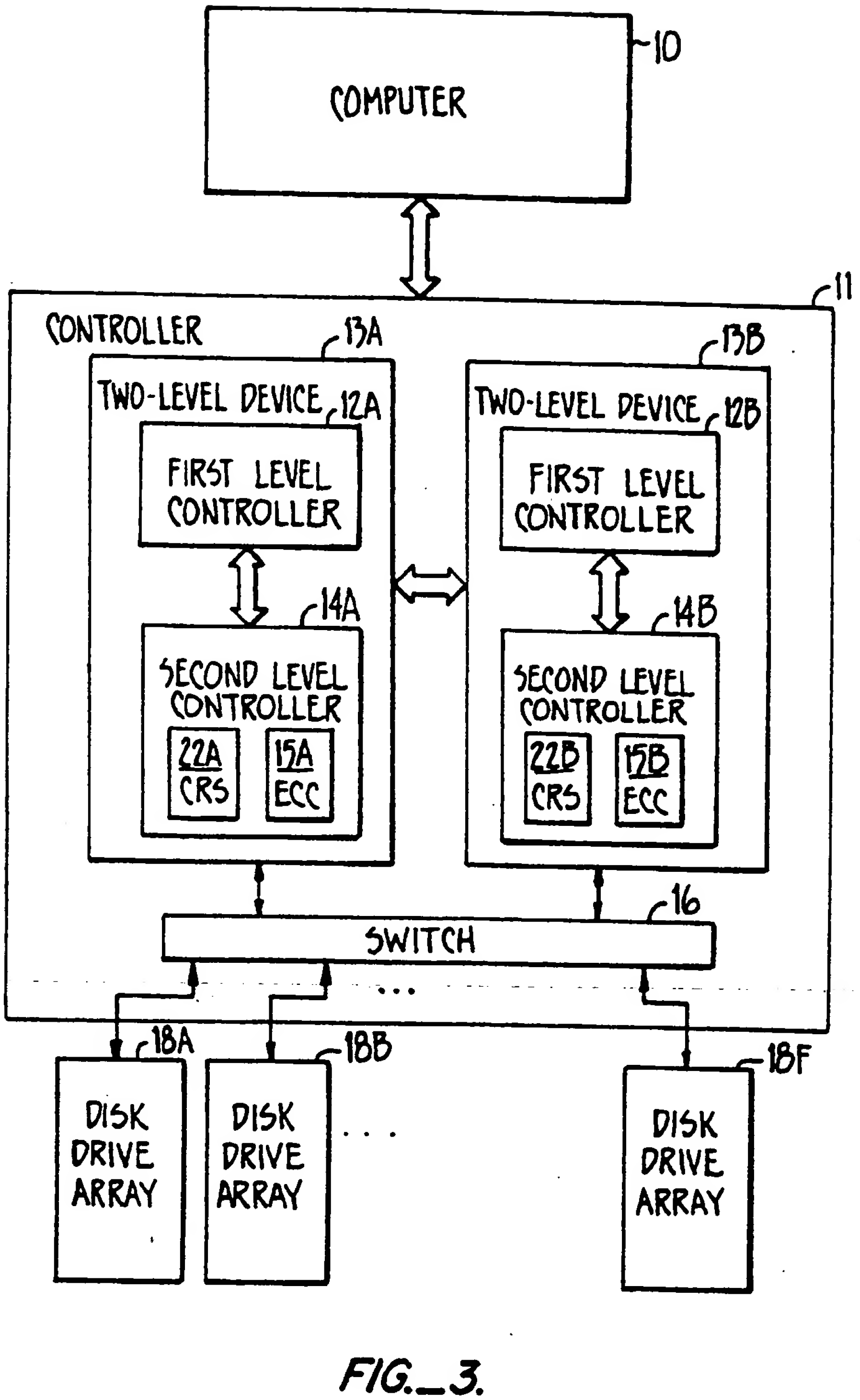

COMPUTER
10
CONTROLLER
11
13A
TWO-LEVEL DEVICE
12A
FIRST LEVEL CONTROLLER
14A
SECOND LEVEL CONTROLLER
22A CRS
15A ECC
13B
TWO-LEVEL DEVICE
12B
FIRST LEVEL CONTROLLER
14B
SECOND LEVEL CONTROLLER
22B CRS
15B ECC
16
SWITCH
18A
18B
18F
DISK DRIVE ARRAY
DISK DRIVE ARRAY
DISK DRIVE ARRAY

FIG._3.

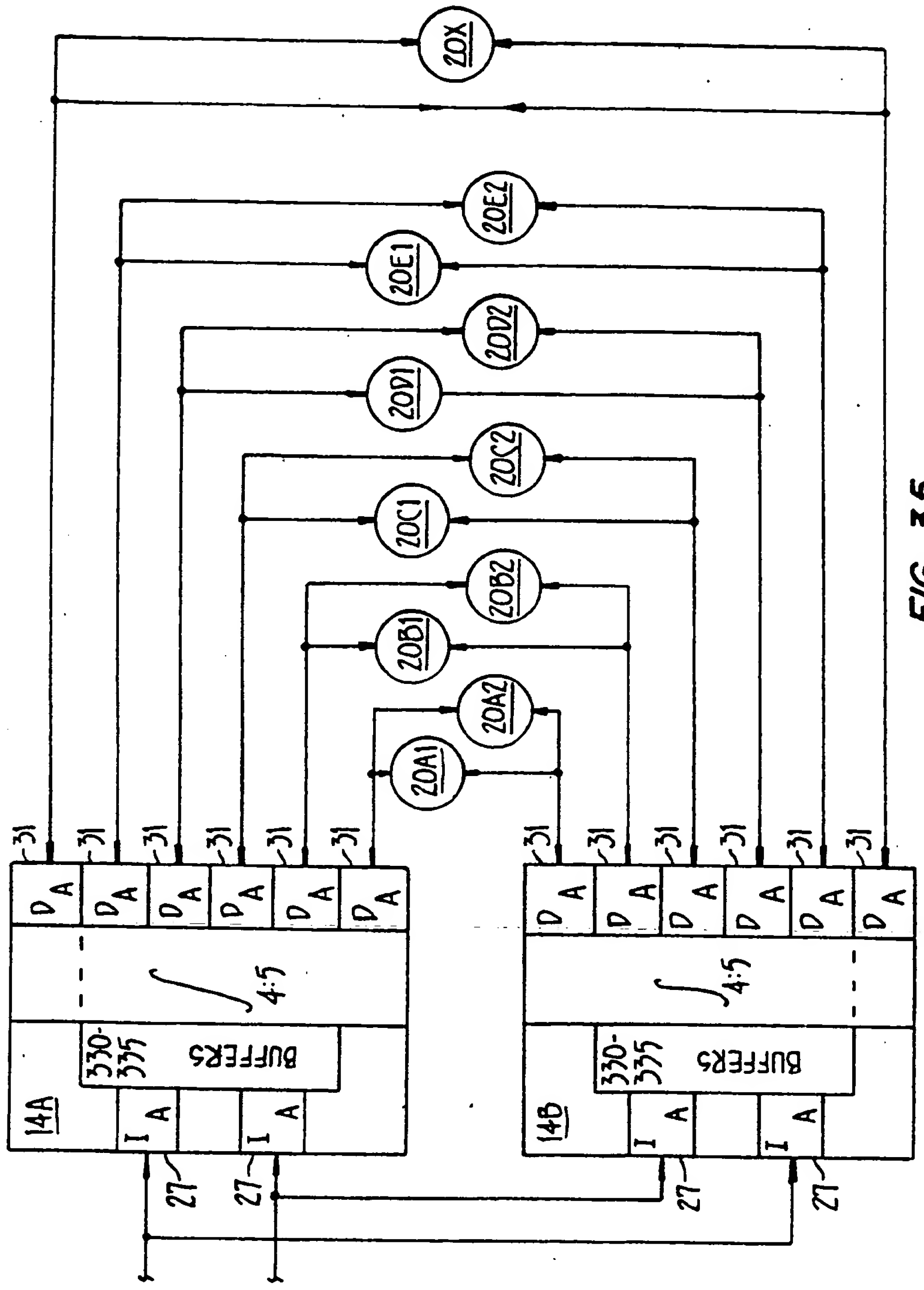
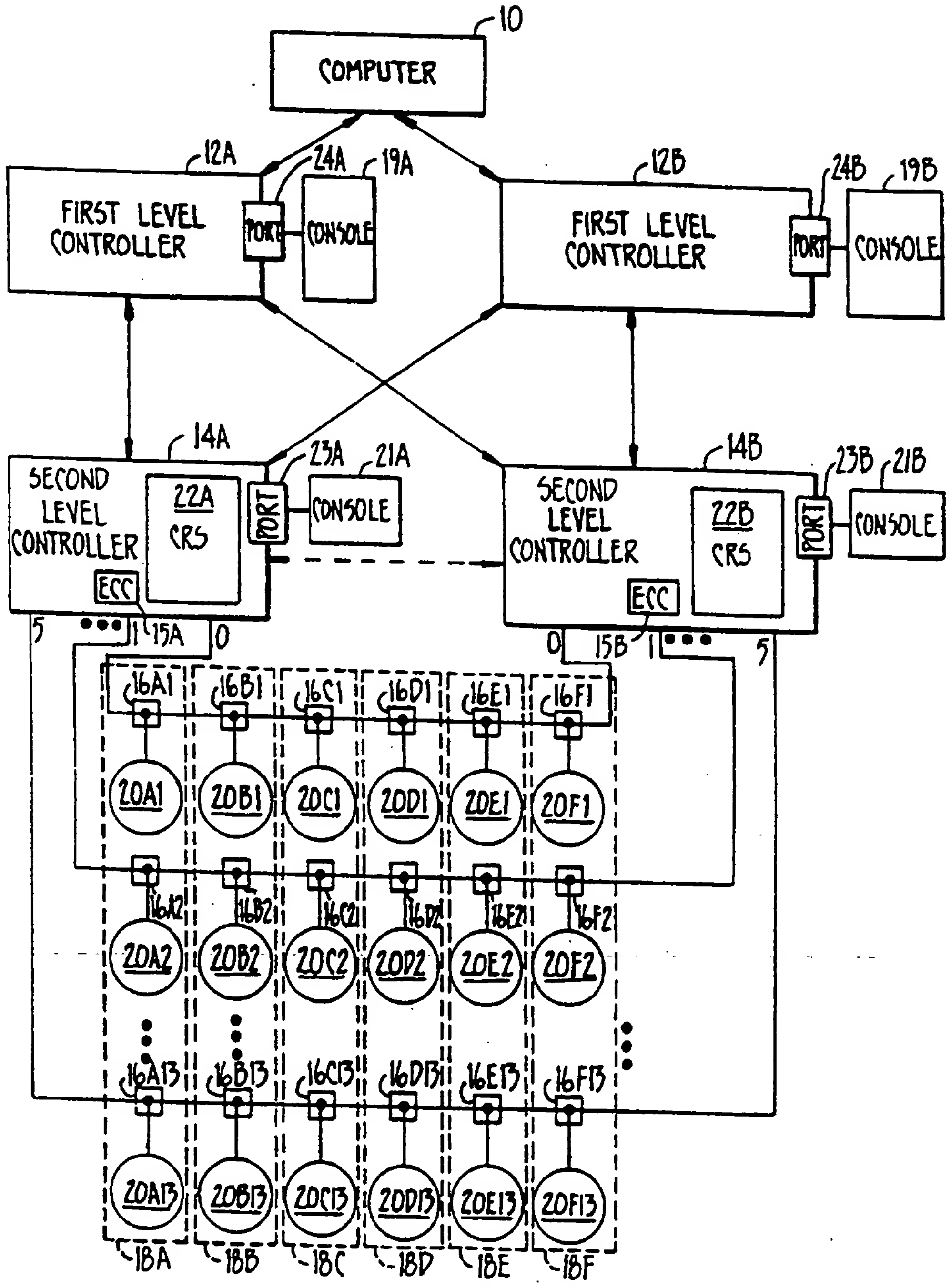

20X
20E2
20E1
20D2
20D1
20C2
20C1
20B2
20B1
20A2
20A1
31
D A
4:5
330-335
BUFFERS
14A
14B
I A
27

FIG. 3.5


10
COMPUTER
12A
FIRST LEVEL CONTROLLER
24A
PORT
19A
CONSOLE
12B
FIRST LEVEL CONTROLLER
24B
PORT
19B
CONSOLE
14A
SECOND LEVEL CONTROLLER
22A
CRS
23A
PORT
21A
CONSOLE
ECC
14B
SECOND LEVEL CONTROLLER
22B
CRS
23B
PORT
21B
CONSOLE
ECC
15A
15B
16A1
16B1
16C1
16D1
16E1
16F1
20A1
20B1
20C1
20D1
20E1
20F1
16A2
16B2
16C2
16D2
16E2
16F2
20A2
20B2
20C2
20D2
20E2
20F2
16A13
16B13
16C13
16D13
16E13
16F13
20A13
20B13
20C13
20D13
20E13
20F13
18A
18B
18C
18D
18E
18F
FIG._4.

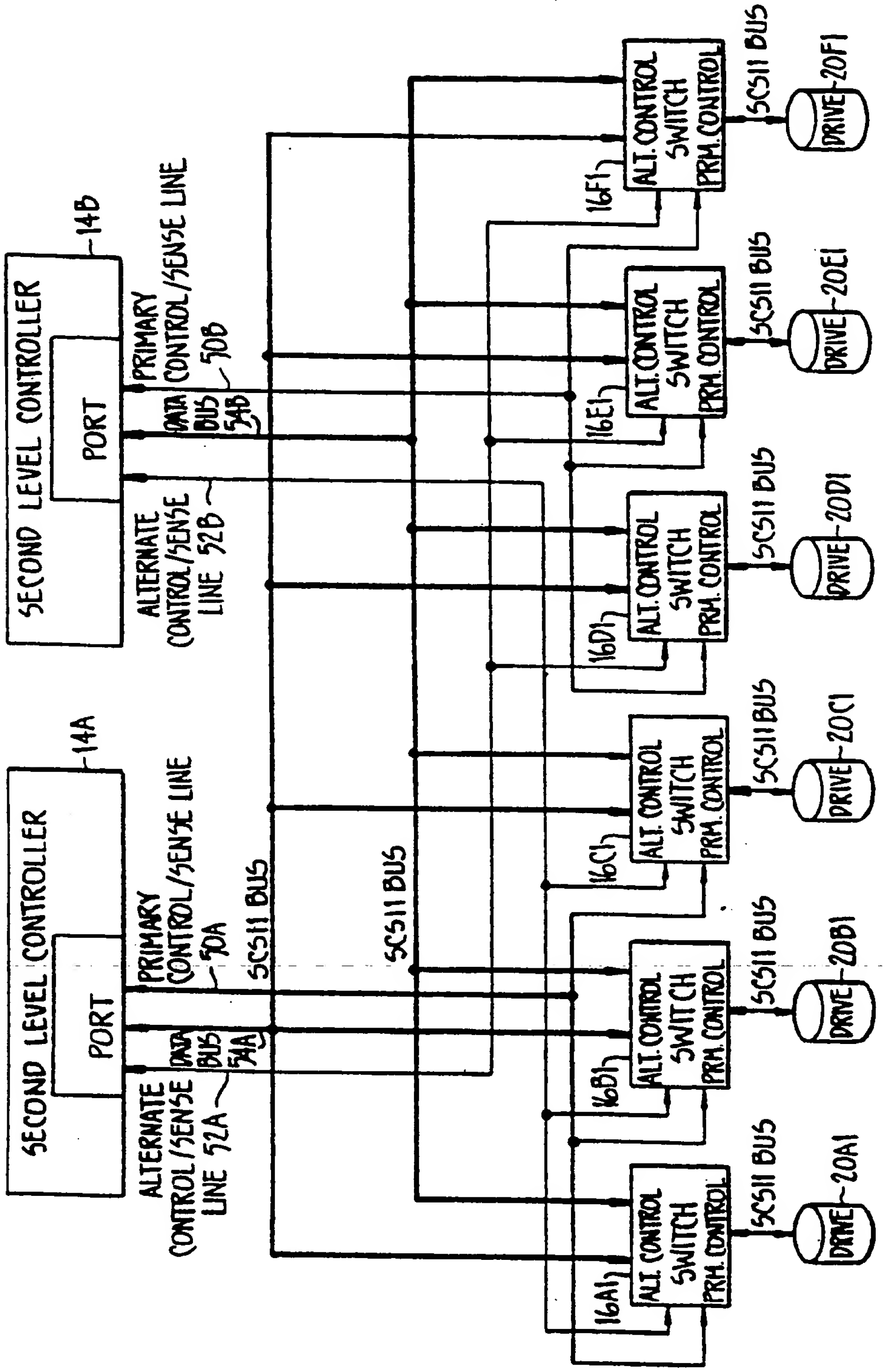

SECOND LEVEL CONTROLLER
PORT
14B
PRIMARY CONTROL/SENSE LINE
50B
DATA BUS
54B
ALTERNATE CONTROL/SENSE LINE 52B
SECOND LEVEL CONTROLLER
PORT
14A
PRIMARY CONTROL/SENSE LINE
50A
SCSI BUS
DATA BUS
54A
ALTERNATE CONTROL/SENSE LINE 52A
SCSI BUS
16A1
ALT. CONTROL SWITCH PRM. CONTROL
SCSI BUS
DRIVE
20A1
16B1
ALT. CONTROL SWITCH PRM. CONTROL
SCSI BUS
DRIVE
20B1
16C1
ALT. CONTROL SWITCH PRM. CONTROL
SCSI BUS
DRIVE
20C1
16D1
ALT. CONTROL SWITCH PRM. CONTROL
SCSI BUS
DRIVE
20D1
16E1
ALT. CONTROL SWITCH PRM. CONTROL
SCSI BUS
DRIVE
20E1
16F1
ALT. CONTROL SWITCH PRM. CONTROL
SCSI BUS
DRIVE
20F1

FIG._5.

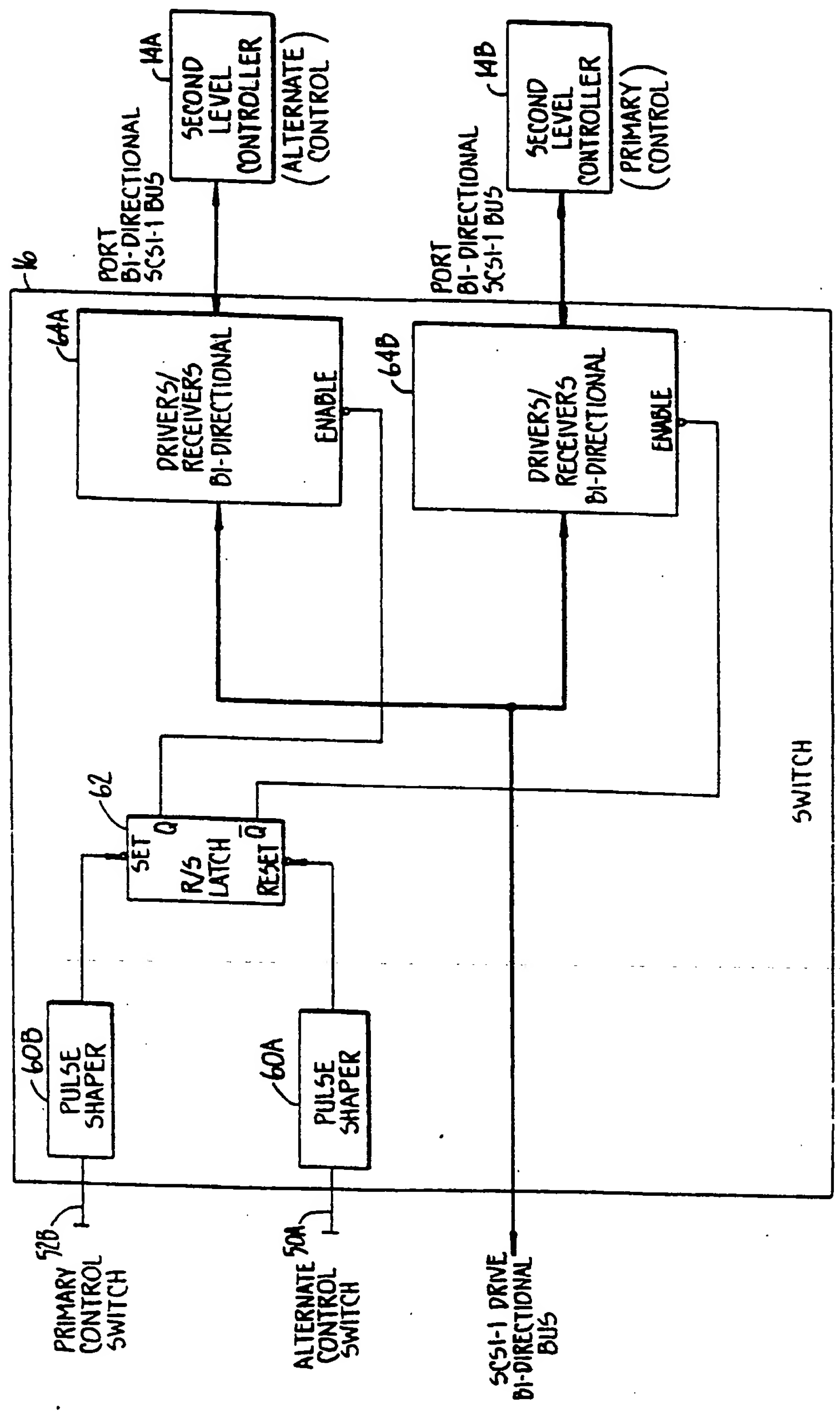

14A
SECOND LEVEL CONTROLLER
(ALTERNATE CONTROL)
14B
SECOND LEVEL CONTROLLER
(PRIMARY CONTROL)
PORT BI-DIRECTIONAL SCSI-1 BUS
PORT BI-DIRECTIONAL SCSI-1 BUS
16
64A
DRIVERS/ RECEIVERS BI-DIRECTIONAL
ENABLE
64B
DRIVERS/ RECEIVERS BI-DIRECTIONAL
ENABLE
62
SET Q
R/S LATCH
RESET $\bar{Q}$
SWITCH
60B
PULSE SHAPER
60A
PULSE SHAPER
52B
PRIMARY CONTROL SWITCH
50A
ALTERNATE CONTROL SWITCH
SCSI-1 DRIVE BI-DIRECTIONAL BUS

FIG._6.

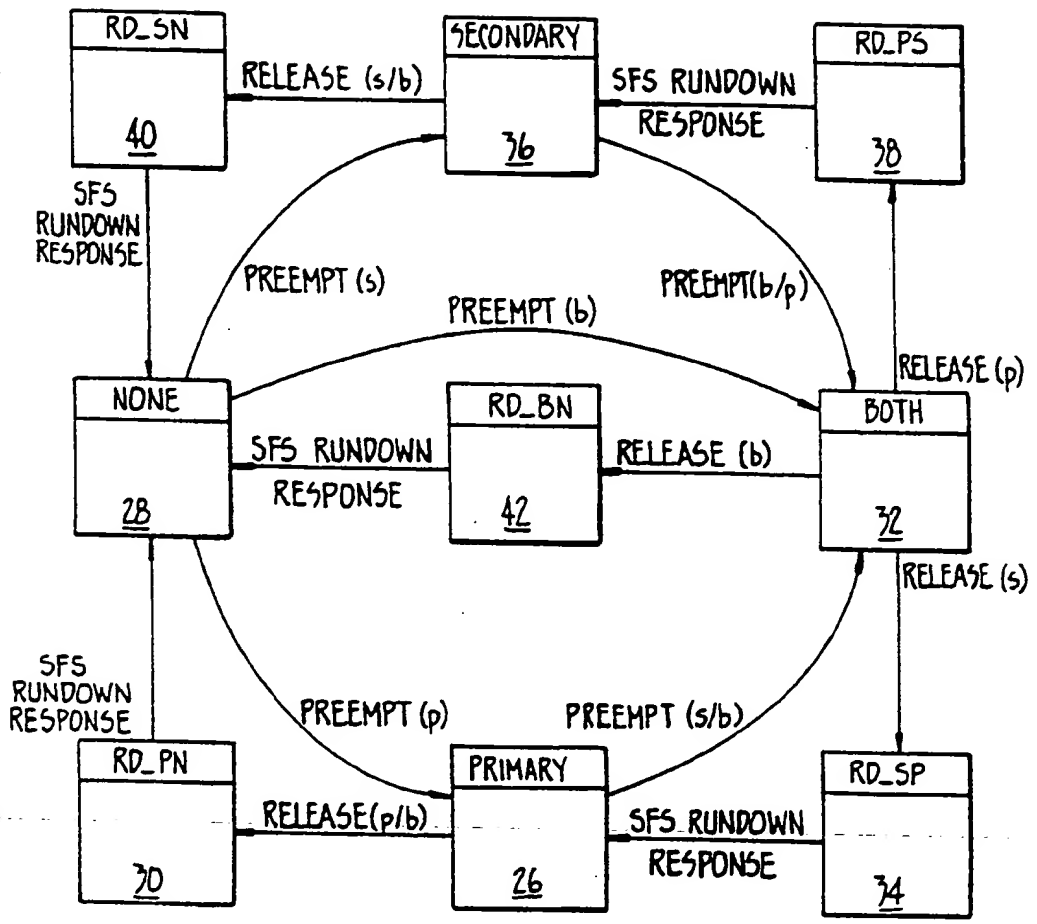

RD_SN
40
RELEASE (s/b)
SECONDARY
36
SFS RUNDOWN RESPONSE
RD_PS
38
SFS RUNDOWN RESPONSE
PREEMPT (s)
PREEMPT(b/p)
PREEMPT (b)
RELEASE (p)
NONE
28
RD_BN
42
SFS RUNDOWN RESPONSE
RELEASE (b)
BOTH
32
RELEASE (s)
SFS RUNDOWN RESPONSE
PREEMPT (p)
PREEMPT (s/b)
RD_PN
30
RELEASE(p/b)
PRIMARY
26
SFS RUNDOWN RESPONSE
RD_SP
34

FIG_7

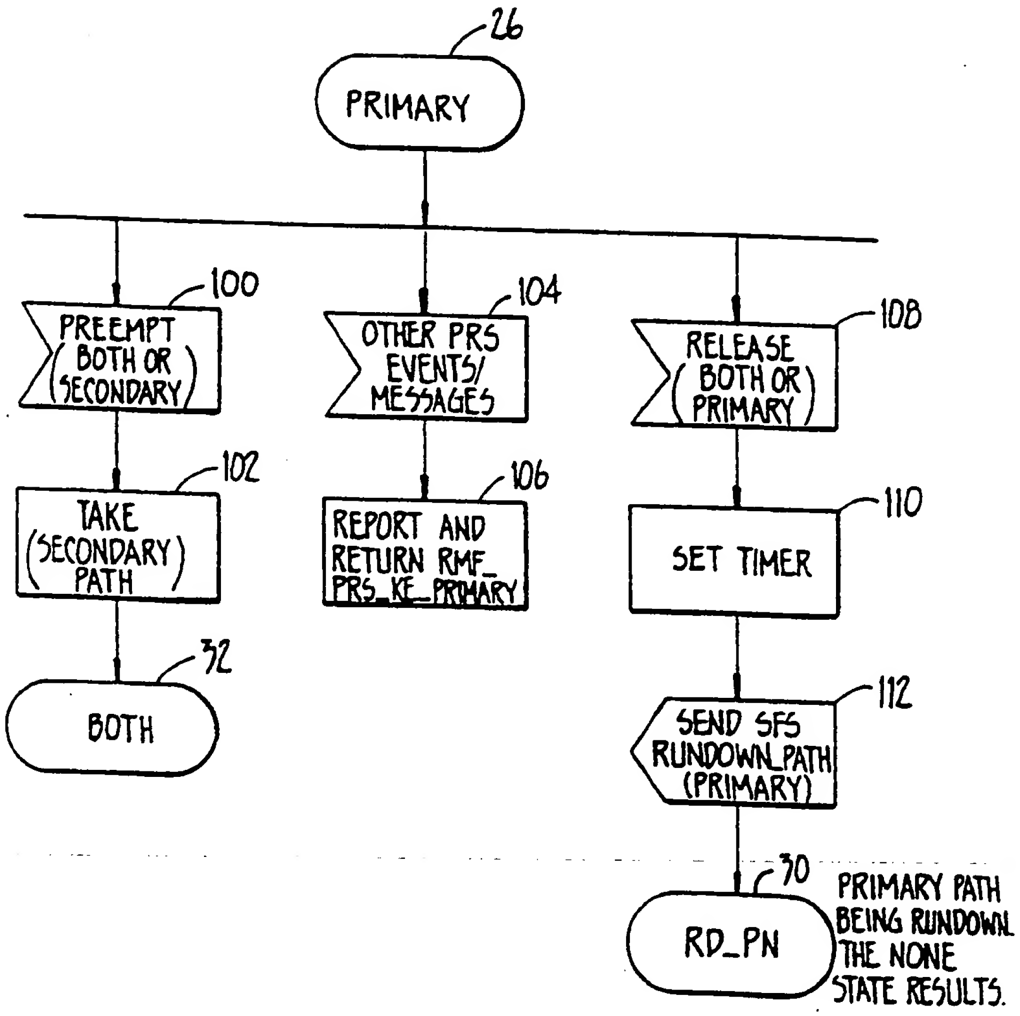

26
PRIMARY
100
PREEMPT (BOTH OR SECONDARY)
104
OTHER PRS EVENTS/ MESSAGES
108
RELEASE (BOTH OR PRIMARY)
102
TAKE (SECONDARY) PATH
106
REPORT AND RETURN RMF_ PRS_KE_PRIMARY
110
SET TIMER
32
BOTH
112
SEND SFS RUNDOWN_PATH (PRIMARY)
30
RD_PN
PRIMARY PATH BEING RUNDOWN THE NONE STATE RESULTS.

FIG_8A.

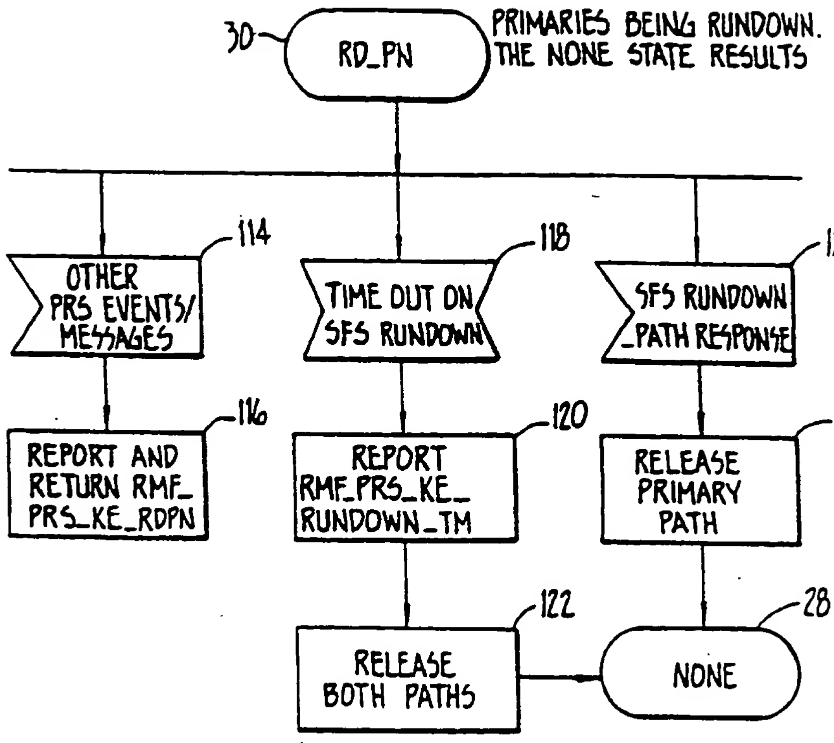

30
RD_PN
PRIMARIES BEING RUNDOWN. THE NONE STATE RESULTS
114
OTHER PRS EVENTS/ MESSAGES
118
TIME OUT ON SFS RUNDOWN
124
SFS RUNDOWN _PATH RESPONSE
116
REPORT AND RETURN RMF_ PRS_KE_RDPN
120
REPORT RMF_PRS_KE_ RUNDOWN_TM
126
RELEASE PRIMARY PATH
122
RELEASE BOTH PATHS
28
NONE

FIG._8B.

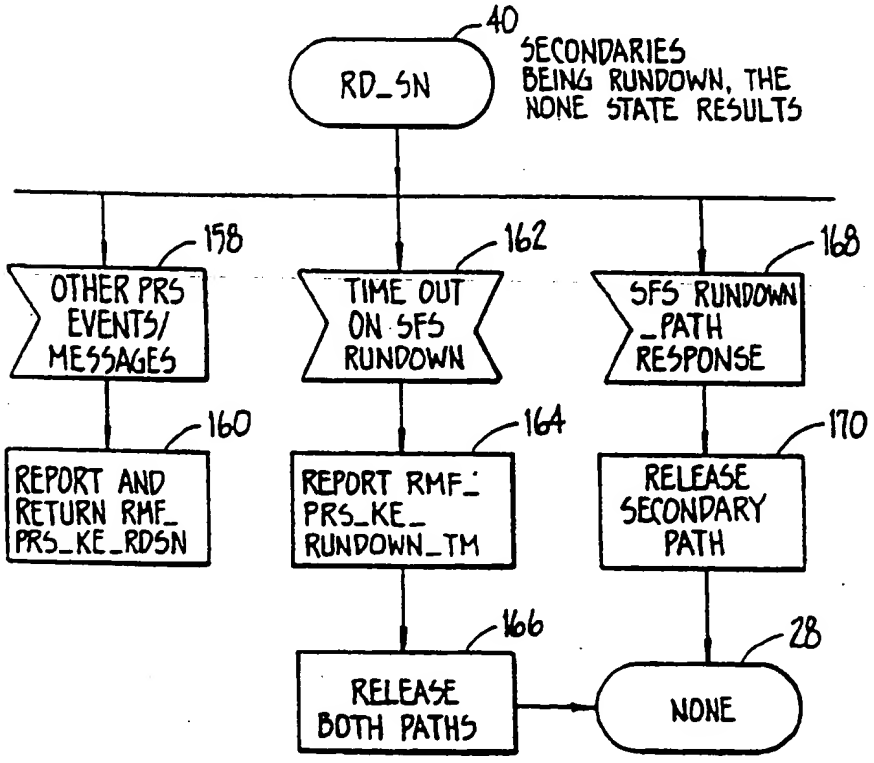

40
RD_SN
SECONDARIES BEING RUNDOWN, THE NONE STATE RESULTS
158
OTHER PRS EVENTS/ MESSAGES
162
TIME OUT ON SFS RUNDOWN
168
SFS RUNDOWN _PATH RESPONSE
160
REPORT AND RETURN RMF_ PRS_KE_RDSN
164
REPORT RMF_ PRS_KE_ RUNDOWN_TM
170
RELEASE SECONDARY PATH
166
RELEASE BOTH PATHS
28
NONE

FIG._8E.

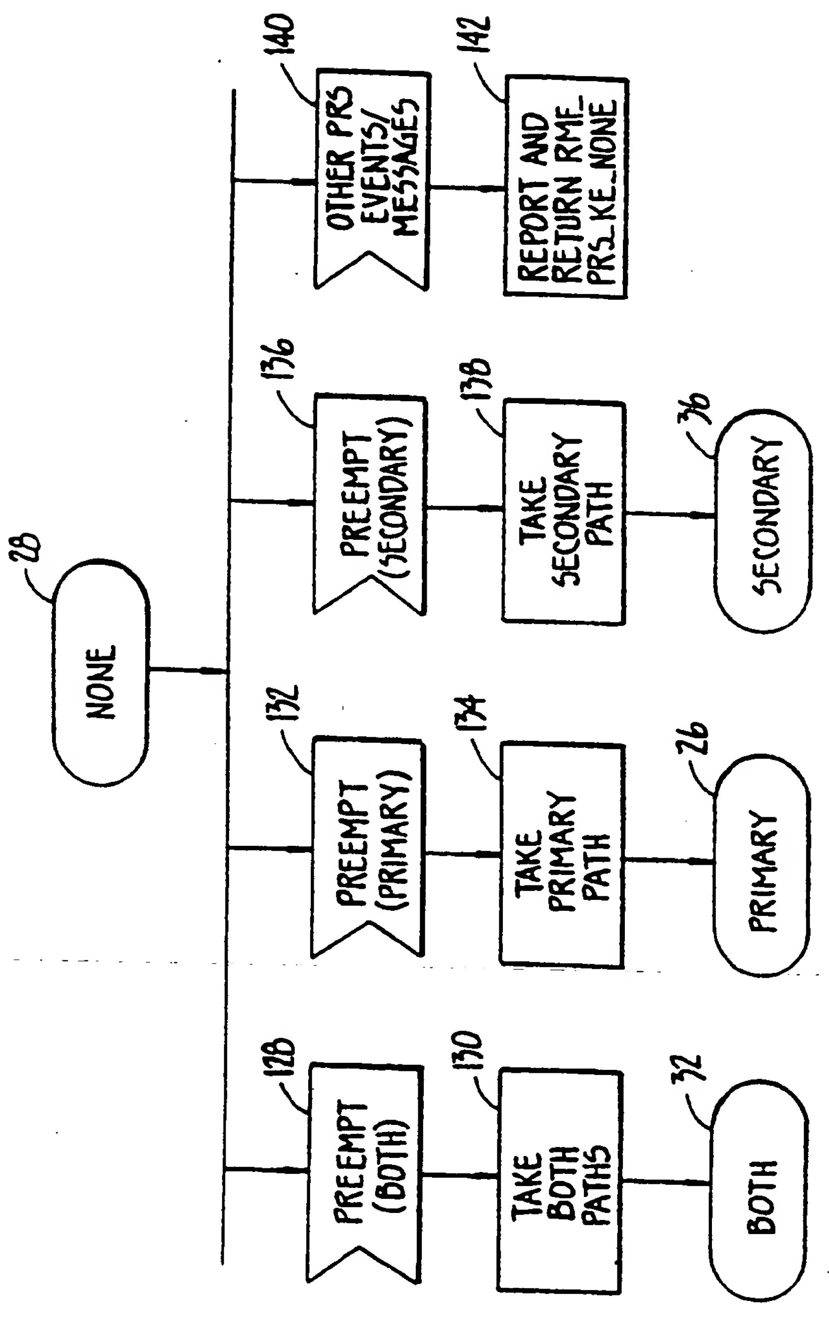

28
NONE
128
PREEMPT (BOTH)
130
TAKE BOTH PATHS
32
BOTH
132
PREEMPT (PRIMARY)
134
TAKE PRIMARY PATH
26
PRIMARY
136
PREEMPT (SECONDARY)
138
TAKE SECONDARY PATH
36
SECONDARY
140
OTHER PRS EVENTS/ MESSAGES
142
REPORT AND RETURN RME_ PRS_KE_NONE

FIG._8C

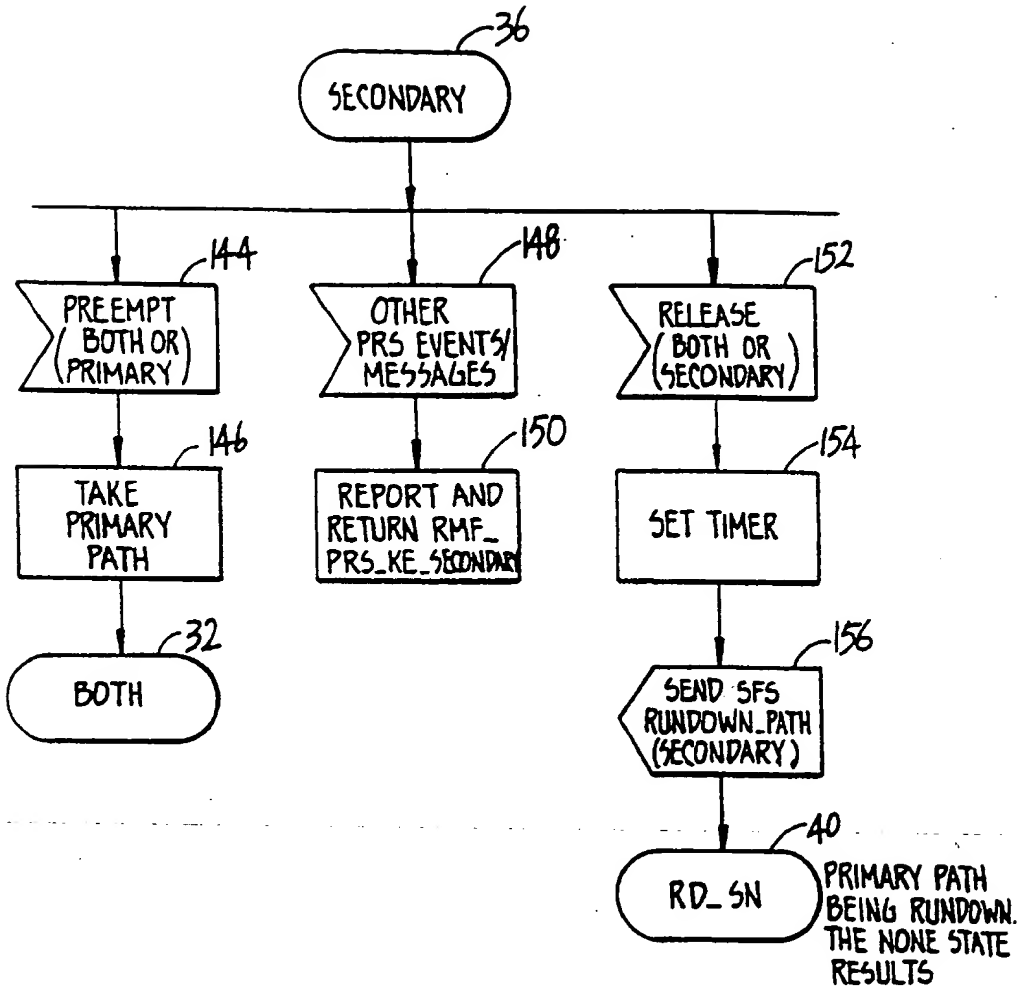

36
SECONDARY
144
PREEMPT (BOTH OR PRIMARY)
148
OTHER PRS EVENTS/ MESSAGES
152
RELEASE (BOTH OR SECONDARY)
146
TAKE PRIMARY PATH
150
REPORT AND RETURN RMF_ PRS_KE_SECONDARY
154
SET TIMER
32
BOTH
156
SEND SFS RUNDOWN_PATH (SECONDARY)
40
RD_SN
PRIMARY PATH BEING RUNDOWN. THE NONE STATE RESULTS

FIG._8D.

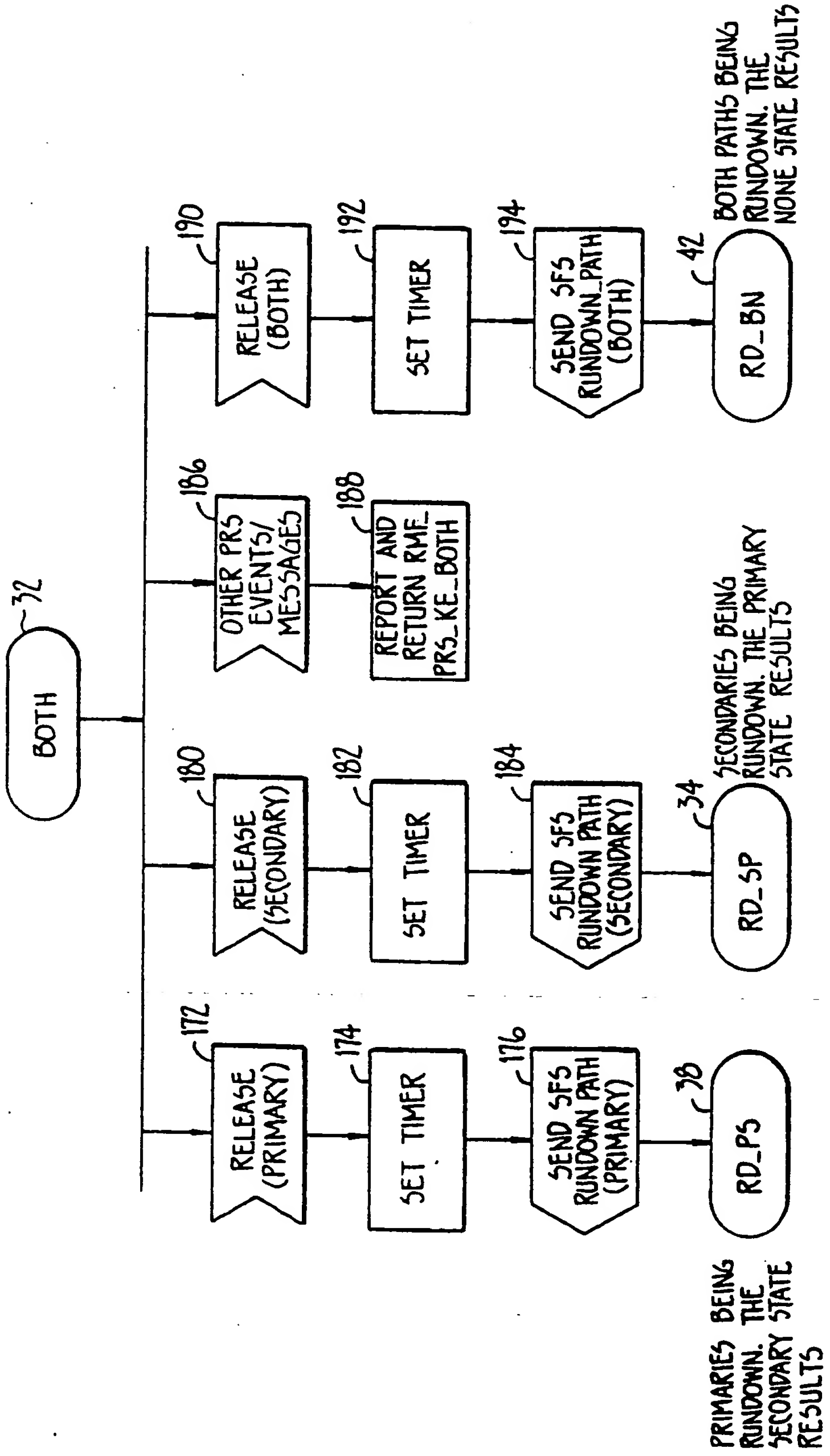

BOTH
32
RELEASE (PRIMARY)
172
SET TIMER
174
SEND SFS RUNDOWN PATH (PRIMARY)
176
RD_PS
38
PRIMARIES BEING RUNDOWN. THE SECONDARY STATE RESULTS
RELEASE (SECONDARY)
180
SET TIMER
182
SEND SFS RUNDOWN PATH (SECONDARY)
184
RD_SP
34
SECONDARIES BEING RUNDOWN. THE PRIMARY STATE RESULTS
OTHER PRS EVENTS/ MESSAGES
186
REPORT AND RETURN RMF_ PRS_KE_BOTH
188
RELEASE (BOTH)
190
SET TIMER
192
SEND SFS RUNDOWN_PATH (BOTH)
194
RD_BN
42
BOTH PATHS BEING RUNDOWN. THE NONE STATE RESULTS

FIG._8F.

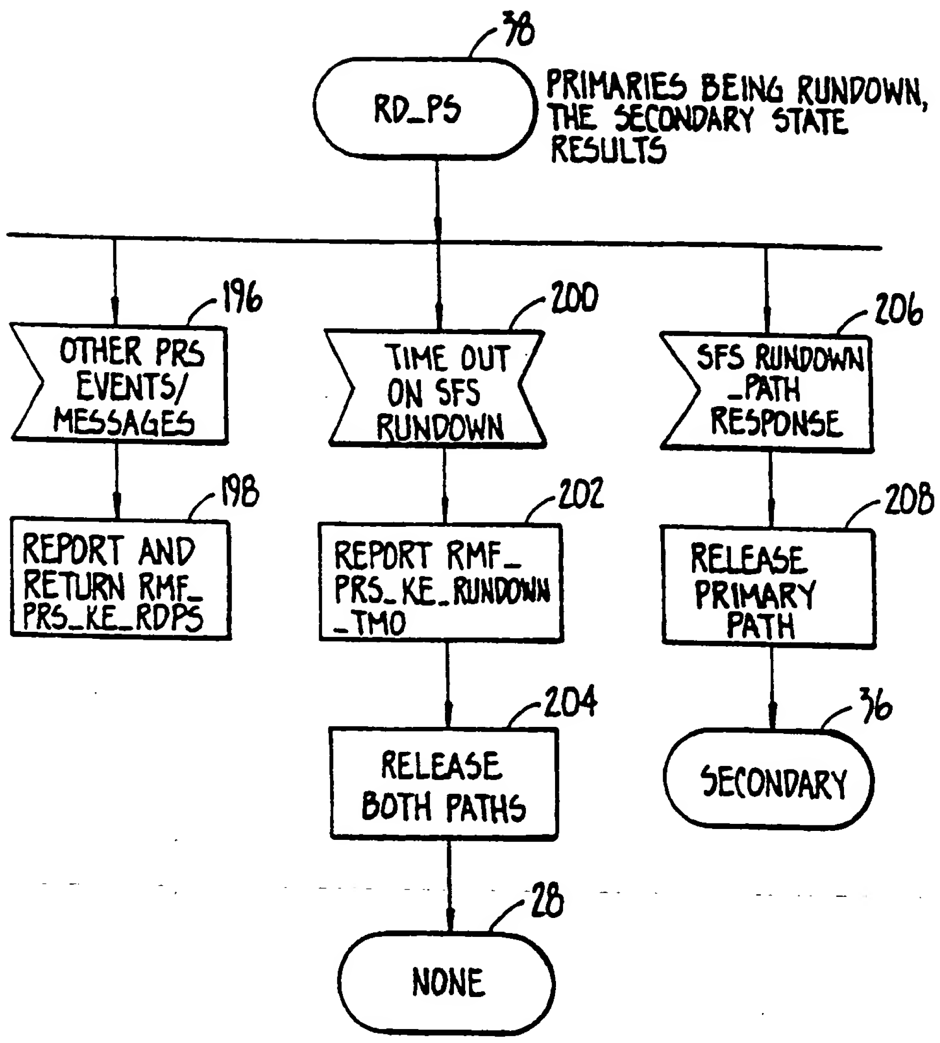

38
RD_PS
PRIMARIES BEING RUNDOWN, THE SECONDARY STATE RESULTS
196
OTHER PRS EVENTS/ MESSAGES
200
TIME OUT ON SFS RUNDOWN
206
SFS RUNDOWN _PATH RESPONSE
198
REPORT AND RETURN RMF_ PRS_KE_RDPS
202
REPORT RMF_ PRS_KE_RUNDOWN _TMO
208
RELEASE PRIMARY PATH
204
RELEASE BOTH PATHS
36
SECONDARY
28
NONE

FIG._8G.

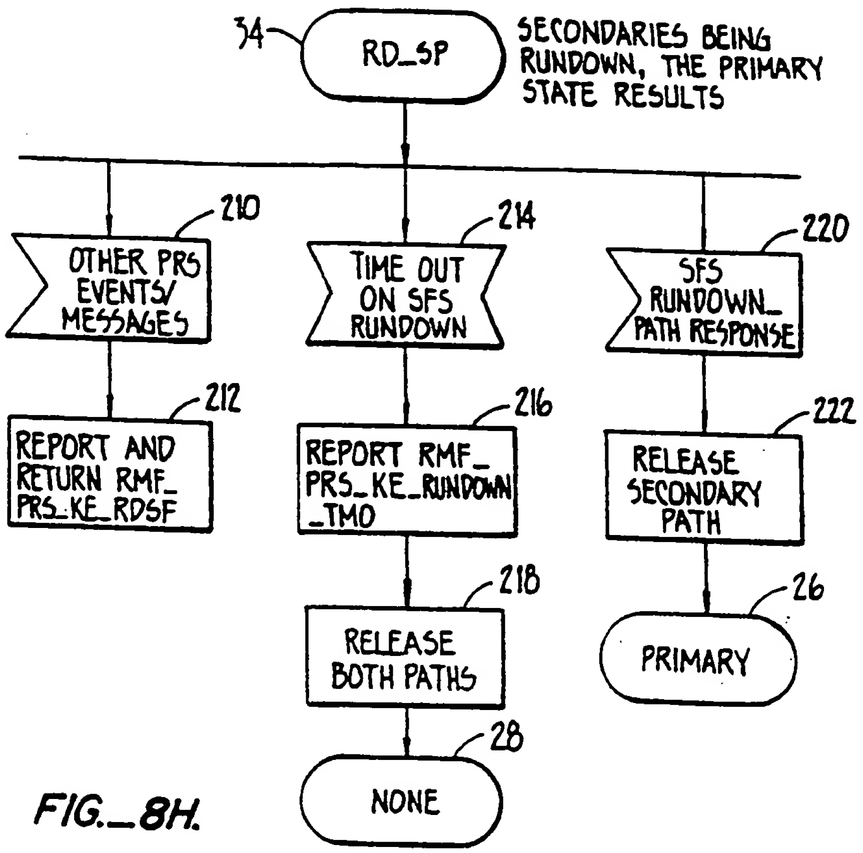

34
RD_SP
SECONDARIES BEING RUNDOWN, THE PRIMARY STATE RESULTS
210
OTHER PRS EVENTS/ MESSAGES
214
TIME OUT ON SFS RUNDOWN
220
SFS RUNDOWN_ PATH RESPONSE
212
REPORT AND RETURN RMF_ PRS_KE_RDSF
216
REPORT RMF_ PRS_KE_RUNDOWN _TMO
222
RELEASE SECONDARY PATH
218
RELEASE BOTH PATHS
26
PRIMARY
28
NONE

FIG._8H.

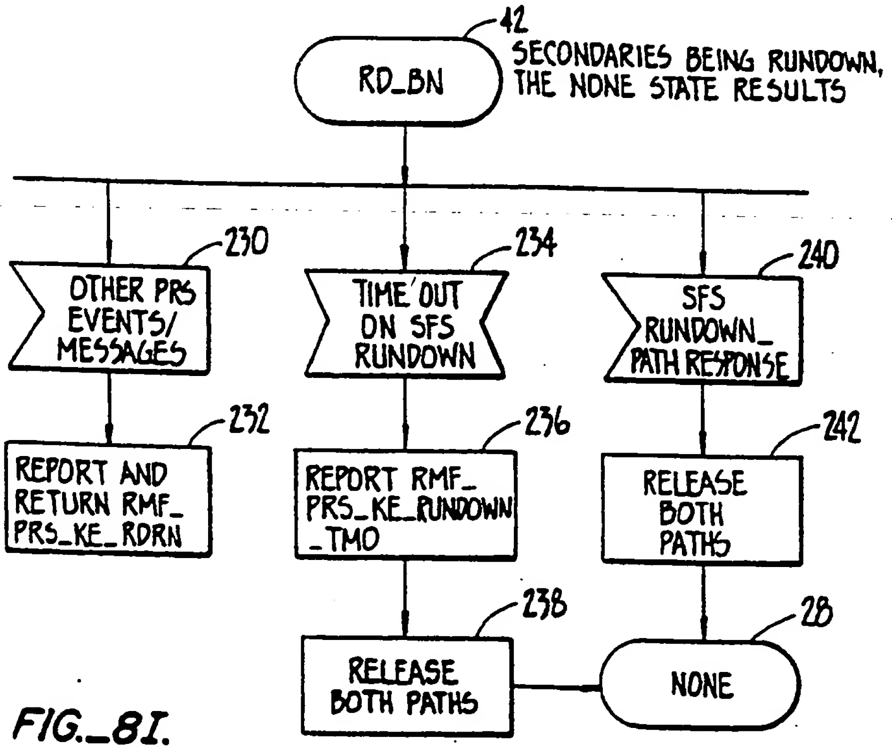

42
RD_BN
SECONDARIES BEING RUNDOWN, THE NONE STATE RESULTS
230
OTHER PRS EVENTS/ MESSAGES
234
TIME OUT ON SFS RUNDOWN
240
SFS RUNDOWN_ PATH RESPONSE
232
REPORT AND RETURN RMF_ PRS_KE_RDRN
236
REPORT RMF_ PRS_KE_RUNDOWN _TMO
242
RELEASE BOTH PATHS
238
RELEASE BOTH PATHS
28
NONE

FIG._8I.

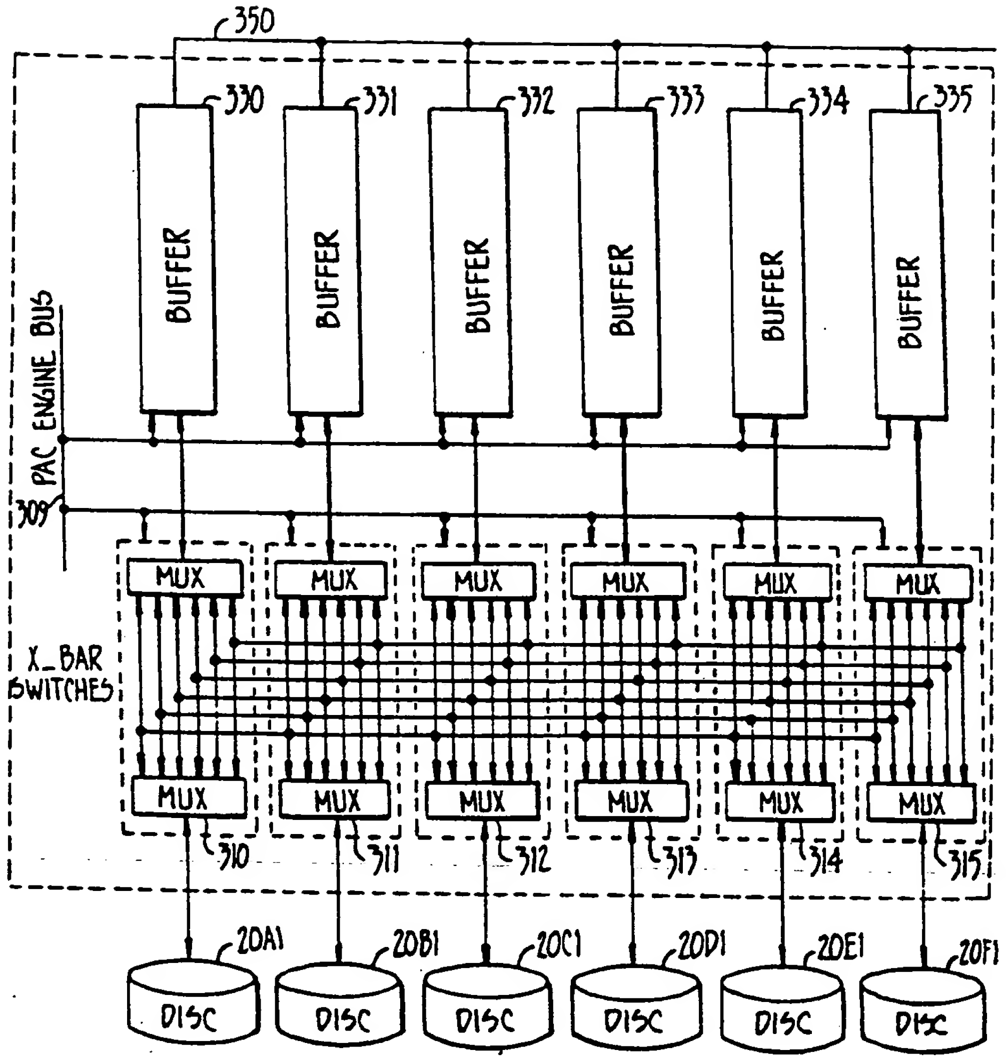

350
330
331
332
333
334
335
BUFFER
BUFFER
BUFFER
BUFFER
BUFFER
BUFFER
PAC ENGINE BUS
309
MUX
X_BAR SWITCHES
310
311
312
313
314
315
20A1
20B1
20C1
20D1
20E1
20F1
DISC

FIG._9.

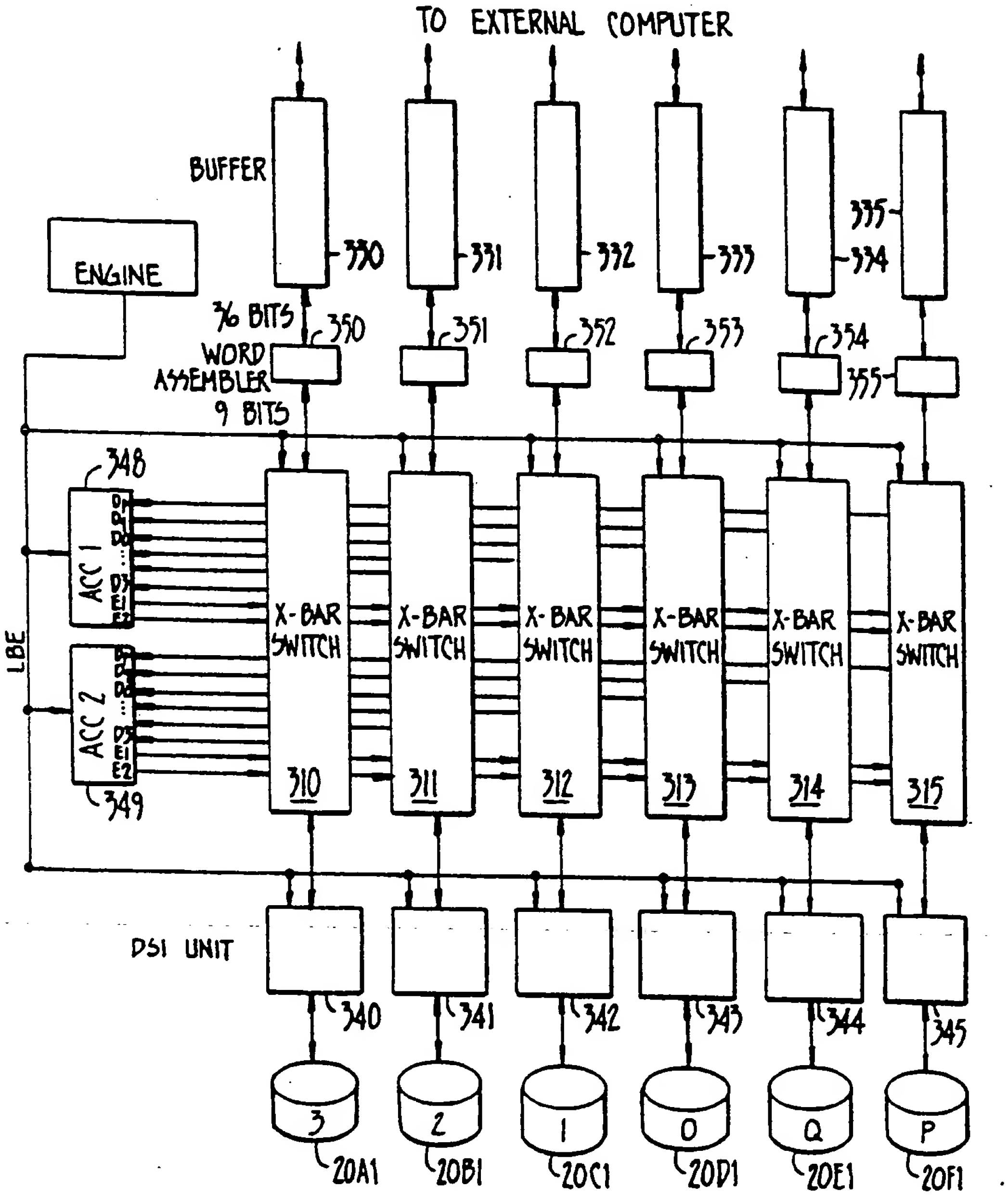

TO EXTERNAL COMPUTER
BUFFER
330
331
332
333
334
335
ENGINE
36 BITS
350
351
352
353
354
355
WORD ASSEMBLER
9 BITS
348
ACC 1
D1
D2
D3
D3
E1
E2
ACC 2
349
LBE
X-BAR SWITCH
310
311
312
313
314
315
DSI UNIT
340
341
342
343
344
345
3
2
1
0
Q
P
20A1
20B1
20C1
20D1
20E1
20F1

FIG._10.

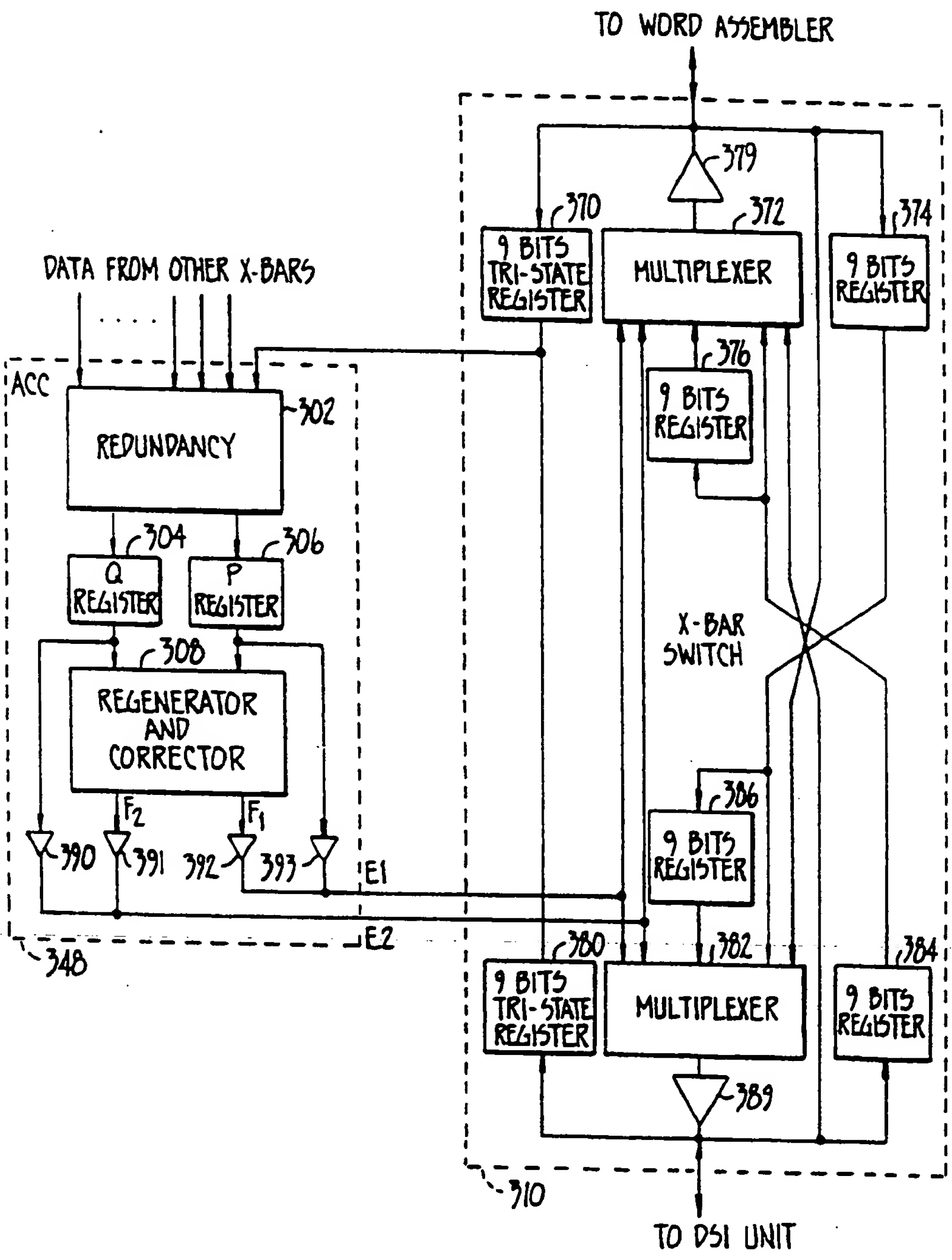

TO WORD ASSEMBLER
379
370
9 BITS TRI-STATE REGISTER
372
MULTIPLEXER
374
9 BITS REGISTER
DATA FROM OTHER X-BARS
376
9 BITS REGISTER
ACC
REDUNDANCY
302
304
306
Q REGISTER
P REGISTER
308
REGENERATOR AND CORRECTOR
X-BAR SWITCH
F_2
F_1
386
9 BITS REGISTER
390
391
392
393
E1
E2
348
380
382
384
9 BITS TRI-STATE REGISTER
MULTIPLEXER
9 BITS REGISTER
389
310
TO DSI UNIT

FIG._11.

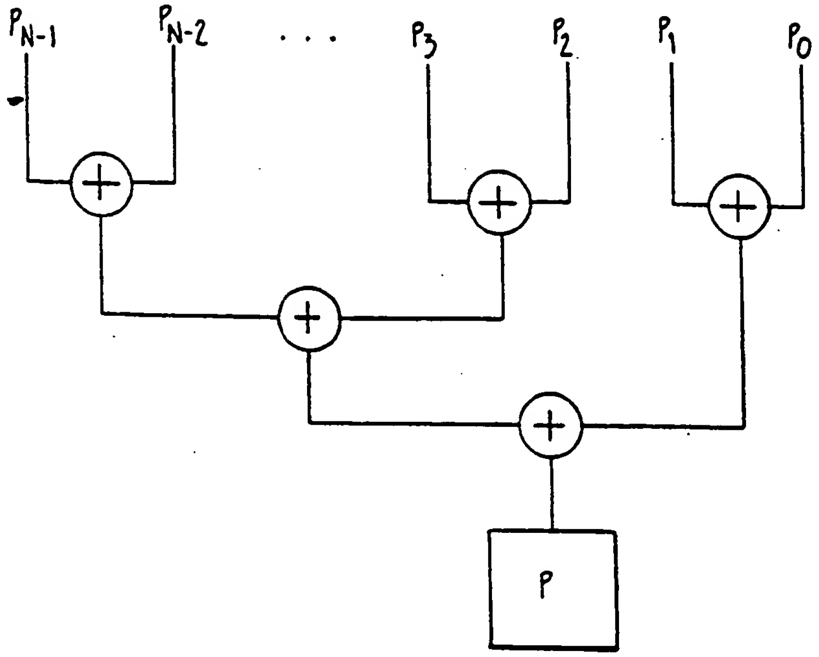


FIG._12A.

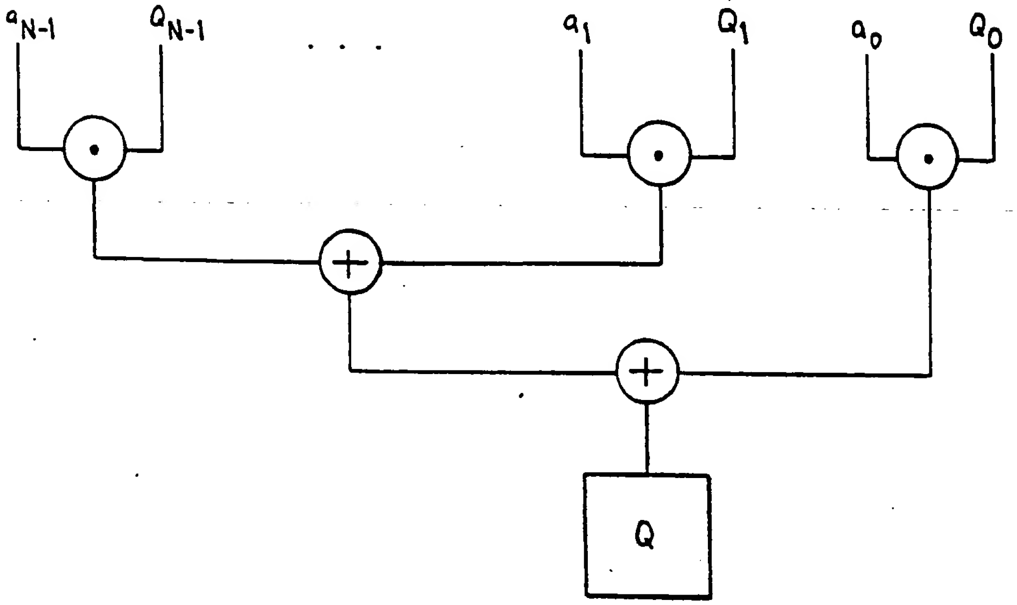


FIG._12B.

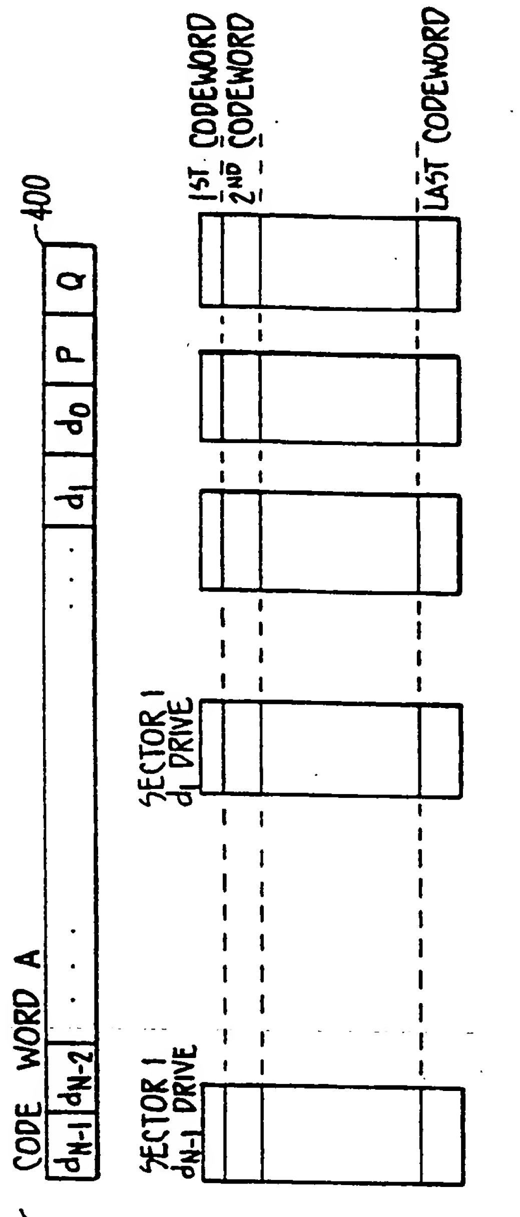


FIG._13A.

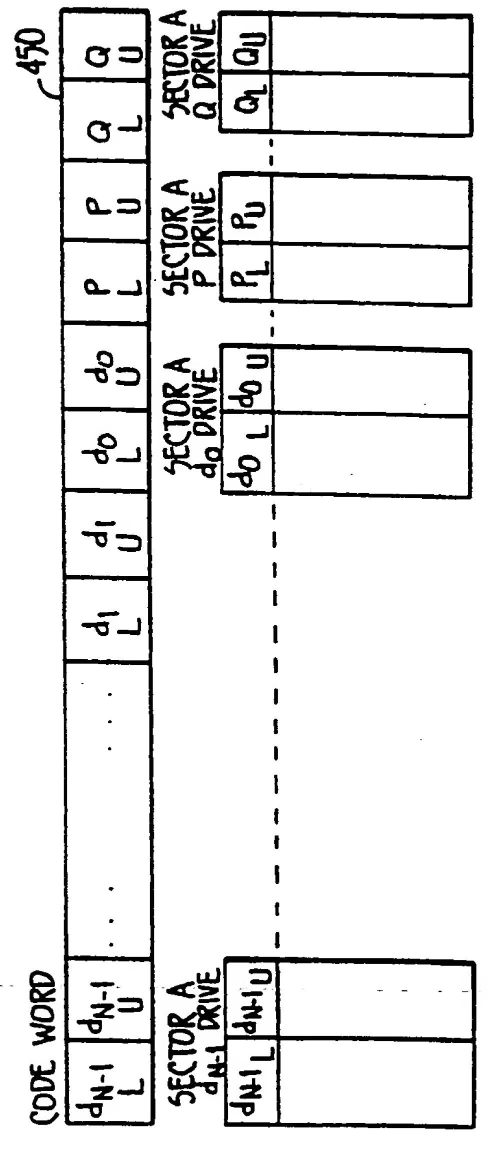


FIG._13B.

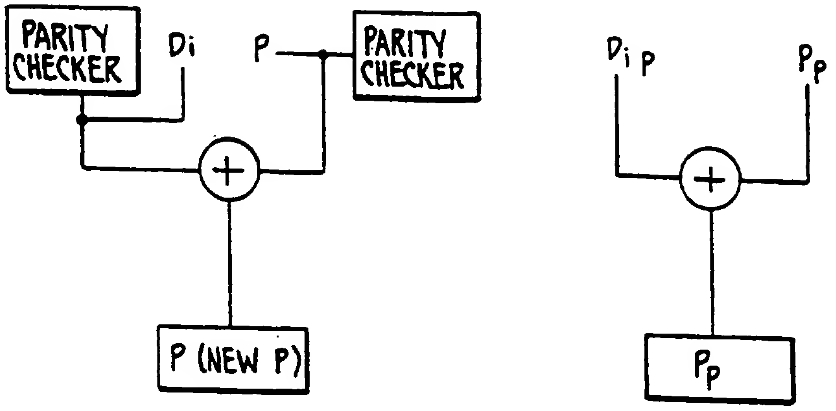


FIG._14A.

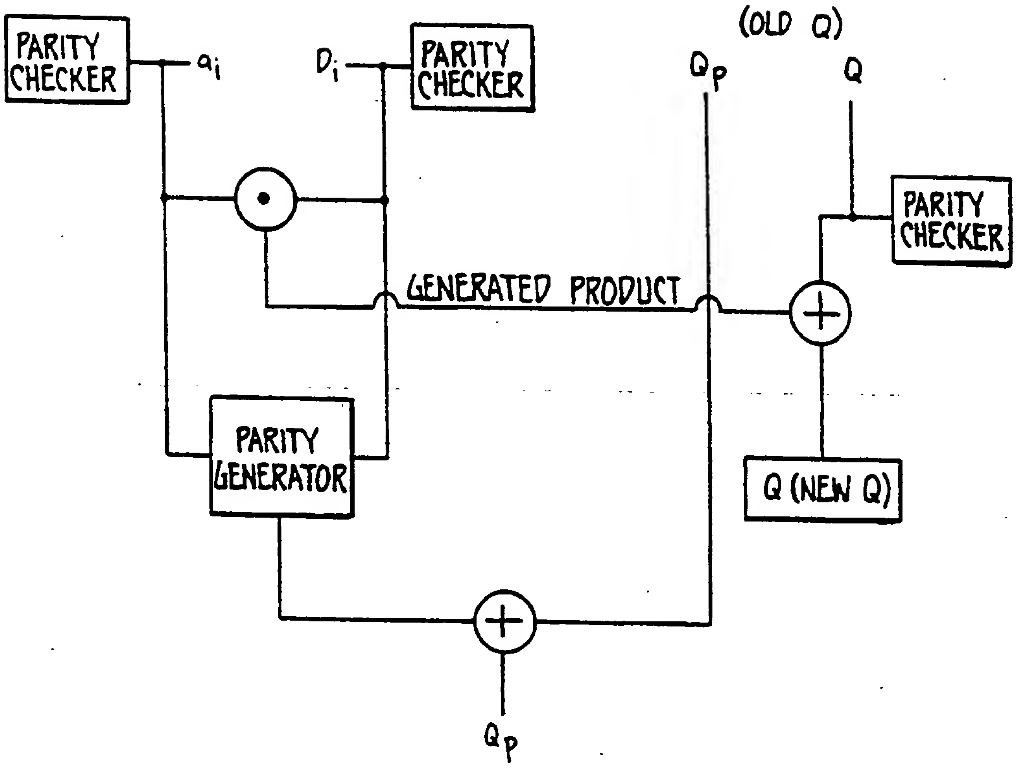


FIG._14B.

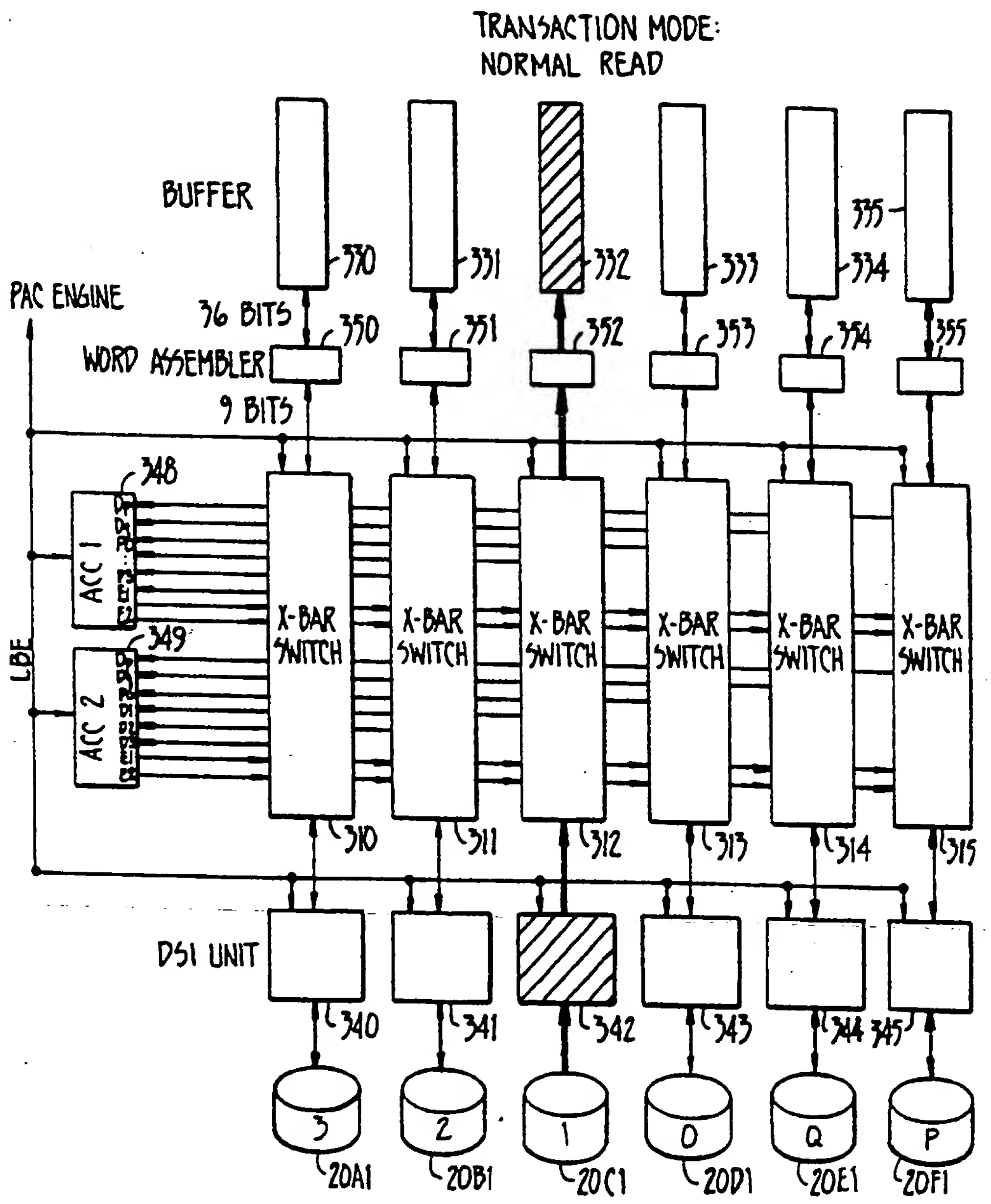

TRANSACTION MODE:
NORMAL READ
BUFFER
330
331
332
333
334
335
PAC ENGINE
36 BITS
WORD ASSEMBLER
350
351
352
353
354
355
9 BITS
348
ACC 1
349
ACC 2
LBE
X-BAR SWITCH
310
311
312
313
314
315
DSI UNIT
340
341
342
343
344
345
3
2
1
0
Q
P
20A1
20B1
20C1
20D1
20E1
20F1

FIG_15.

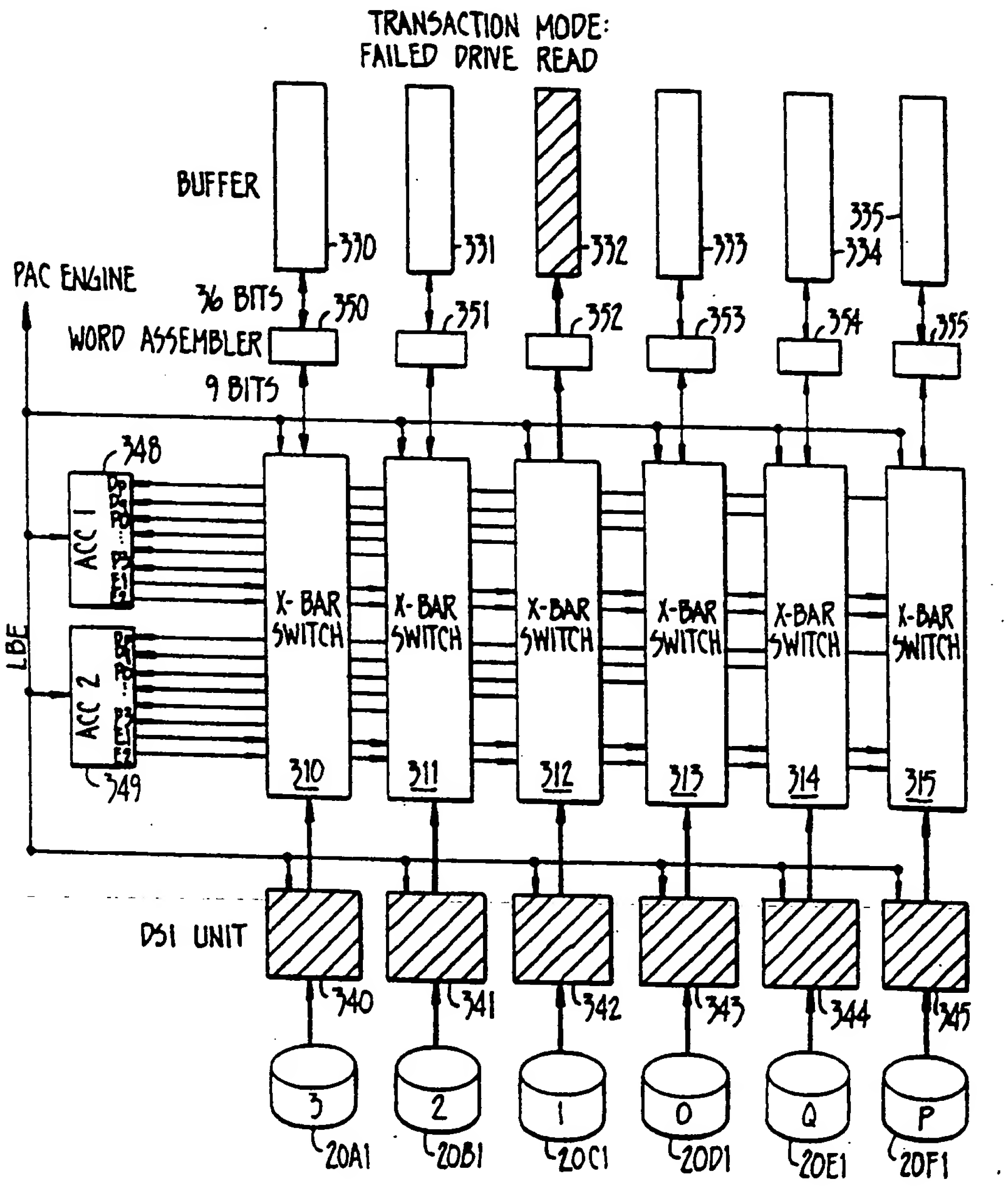

TRANSACTION MODE:
FAILED DRIVE READ
BUFFER
330
331
332
333
334
335
PAC ENGINE
36 BITS
WORD ASSEMBLER
9 BITS
350
351
352
353
354
355
348
ACC 1
ACC 2
349
LBE
X-BAR SWITCH
310
311
312
313
314
315
DSI UNIT
340
341
342
343
344
345
3
2
1
0
Q
P
20A1
20B1
20C1
20D1
20E1
20F1

FIG._16.

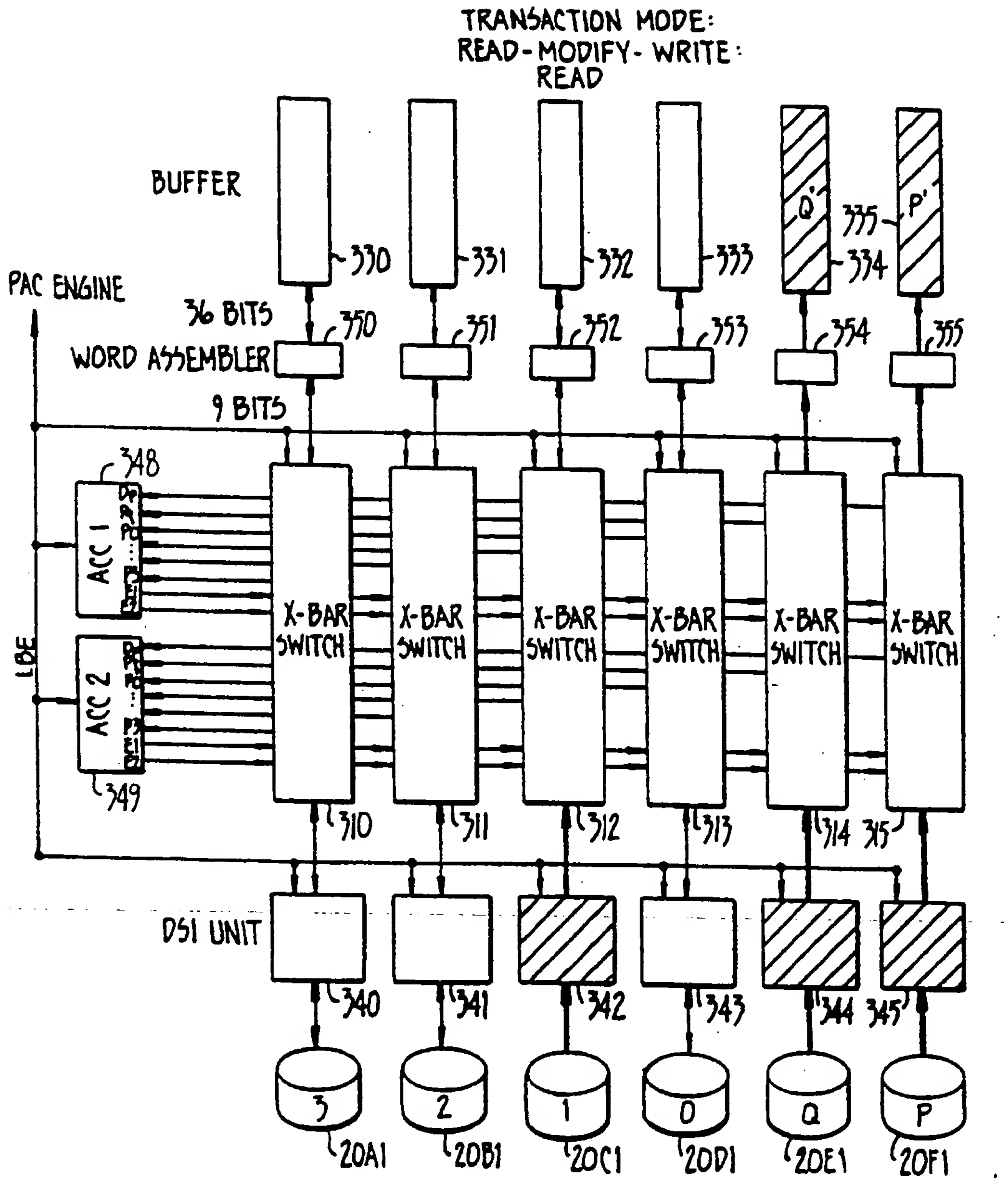

TRANSACTION MODE:
READ-MODIFY-WRITE:
READ
BUFFER
Q'
P'
330
331
332
333
334
335
PAC ENGINE
36 BITS
WORD ASSEMBLER
350
351
352
353
354
355
9 BITS
348
ACC 1
ACC 2
349
LBE
X-BAR SWITCH
310
311
312
313
314
315
DSI UNIT
340
341
342
343
344
345
3
2
1
0
Q
P
20A1
20B1
20C1
20D1
20E1
20F1

FIG._17.

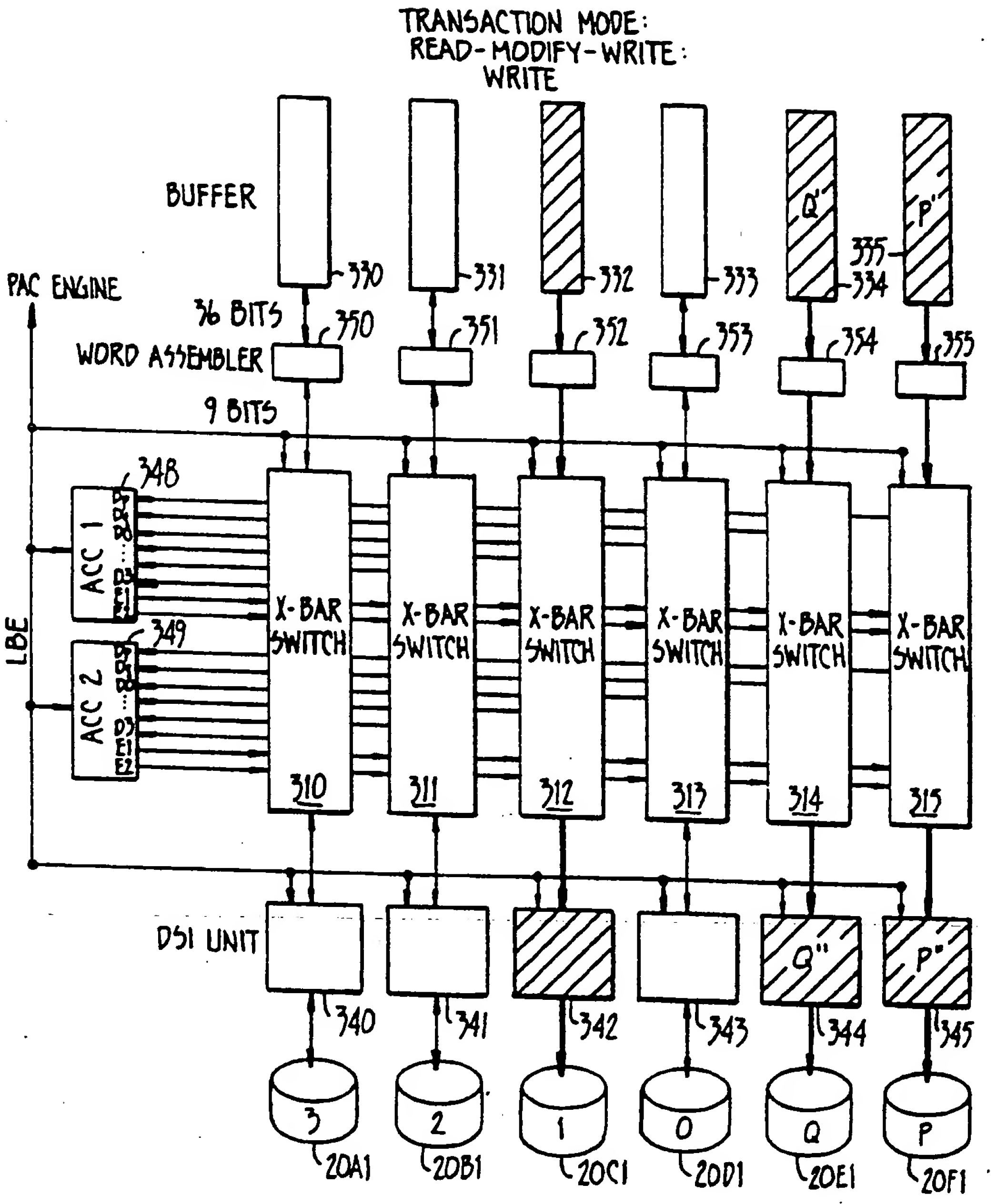

TRANSACTION MODE:
READ-MODIFY-WRITE:
WRITE
BUFFER
PAC ENGINE
36 BITS
WORD ASSEMBLER
9 BITS
LBE
ACC 1
ACC 2
348
349
Q'
P'
330
331
332
333
334
335
350
351
352
353
354
355
X-BAR SWITCH
310
311
312
313
314
315
DSI UNIT
Q''
P''
340
341
342
343
344
345
3
2
1
0
Q
P
20A1
20B1
20C1
20D1
20E1
20F1

FIG._18.

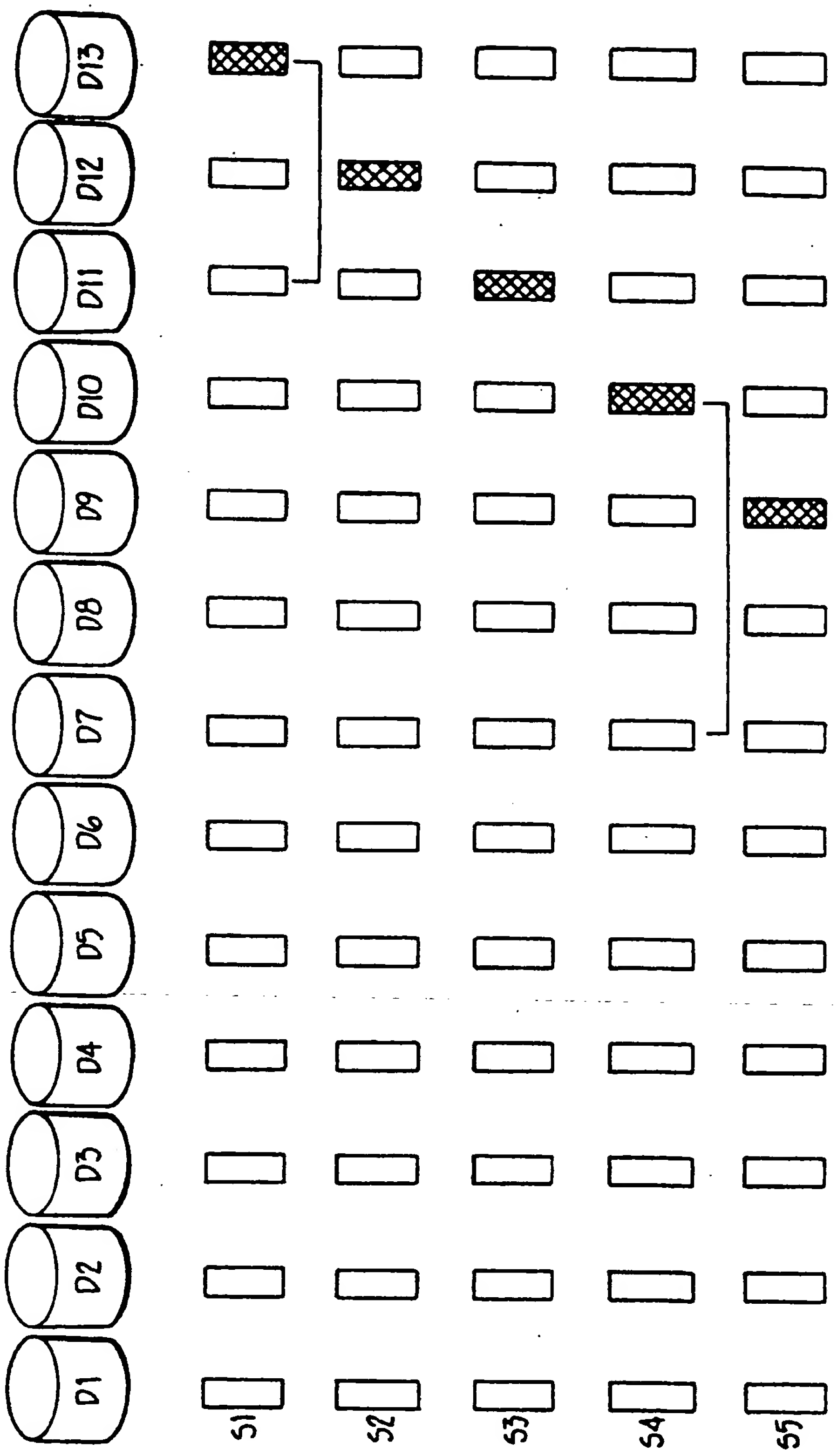


FIG._19.

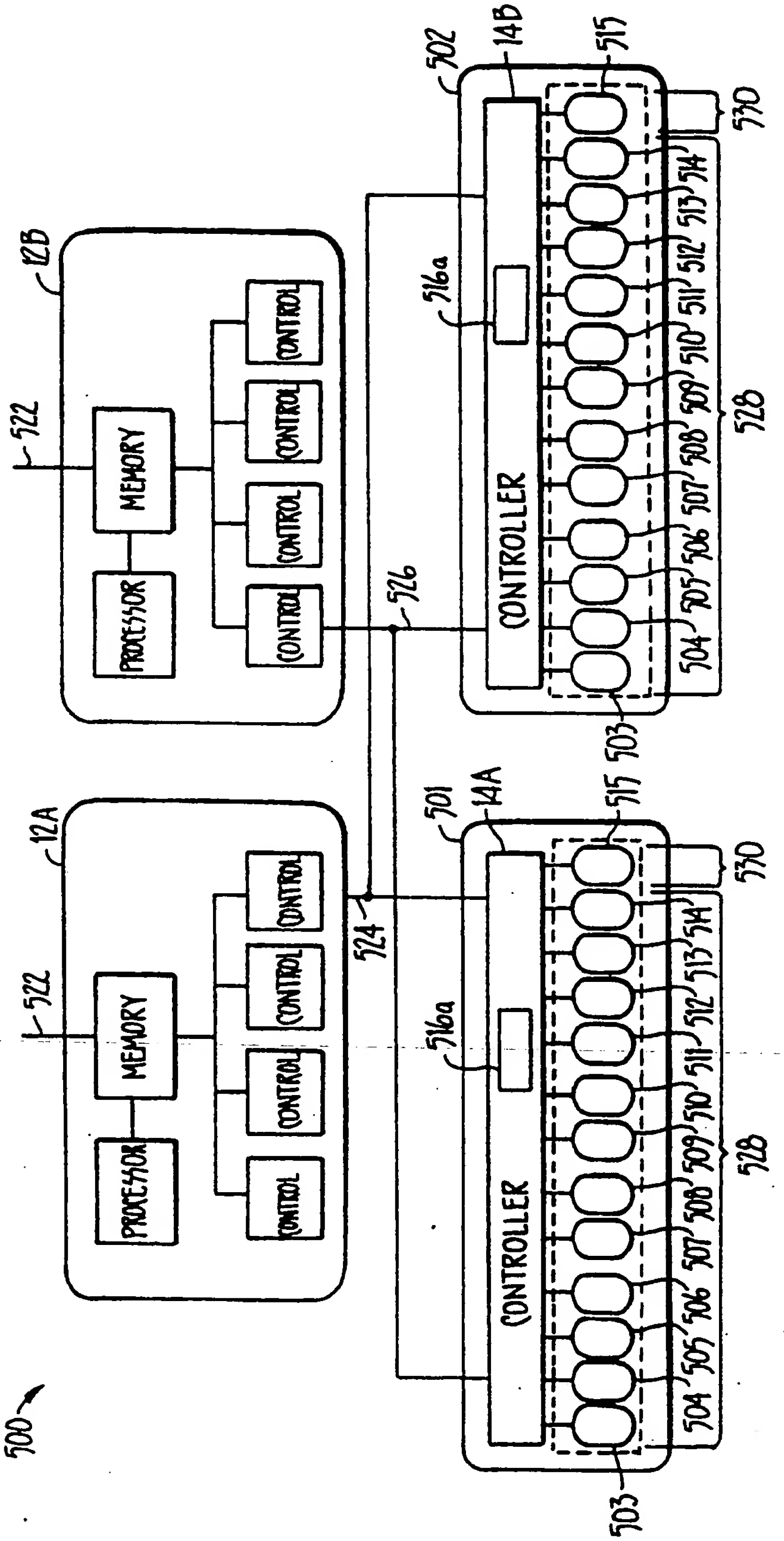

500
12A
12B
522
PROCESSOR
MEMORY
CONTROL
524
526
501
502
14A
14B
516a
CONTROLLER
503
504
505
506
507
508
509
510
511
512
513
514
515
528
530

FIG. 20.

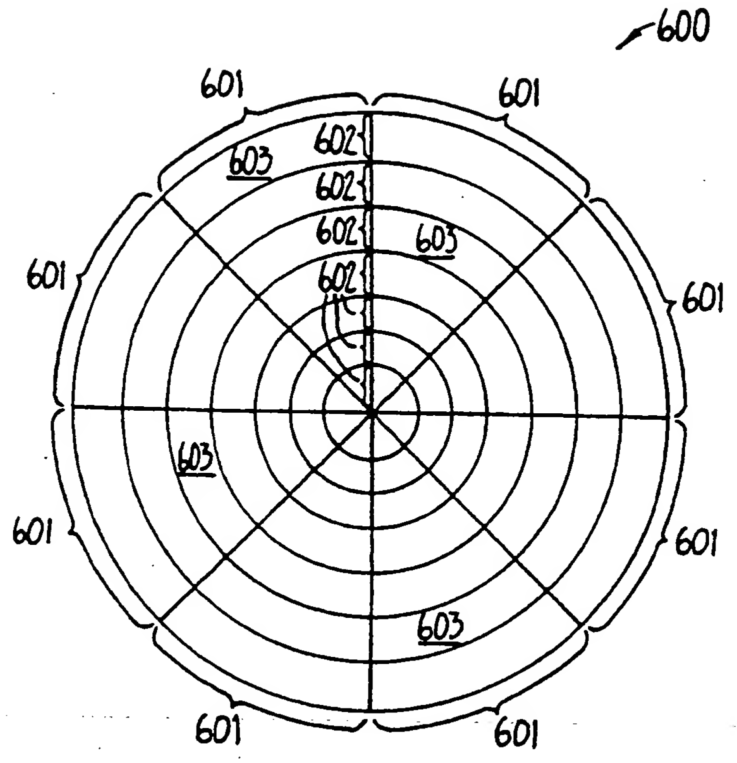

600
601
601
601
601
601
601
601
601
602
602
602
602
603
603
603
603

FIG. 21.

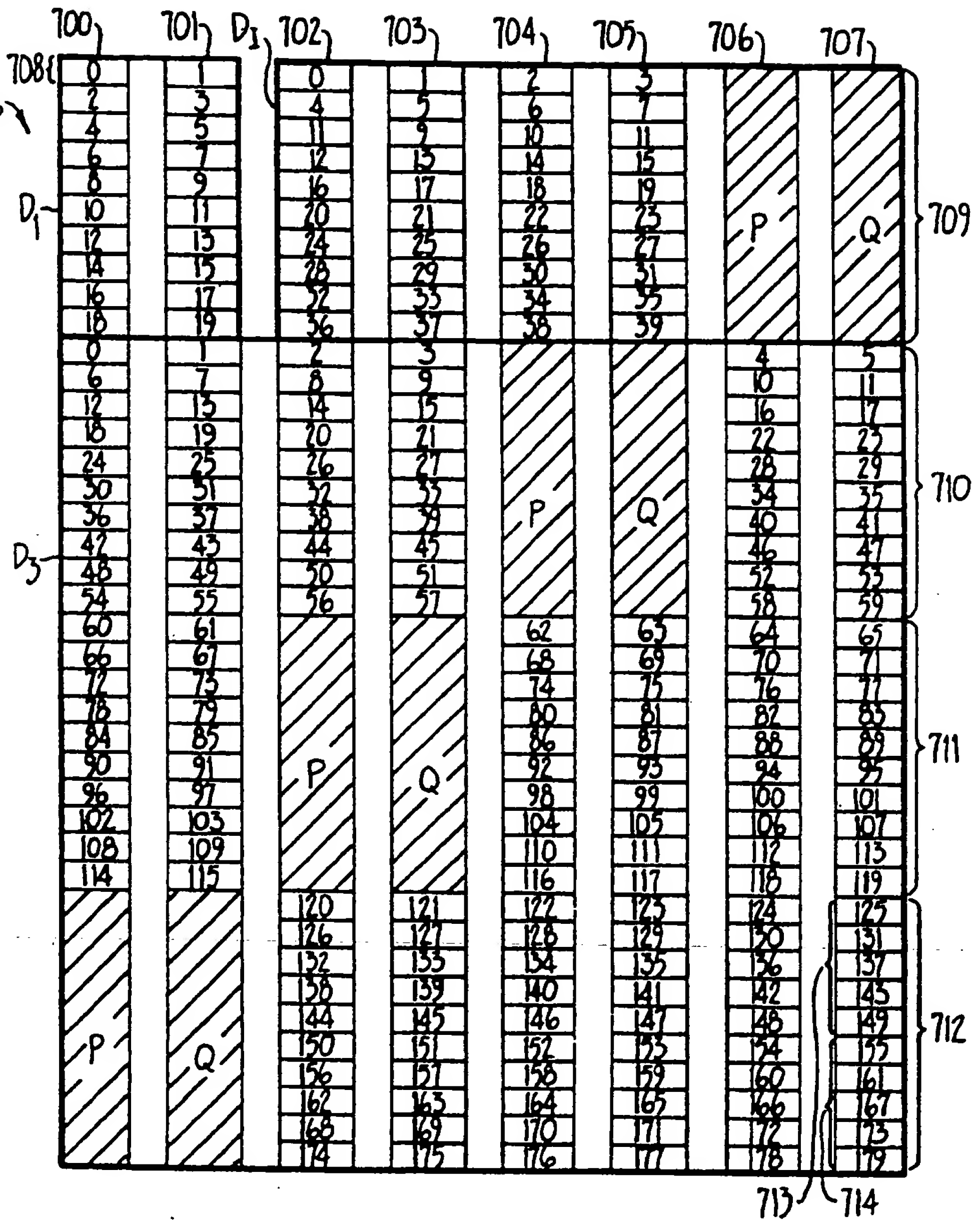

700
701
702
703
704
705
706
707
708
716
709
710
711
712
713
714
D1
D3
P
Q

FIG. 22.

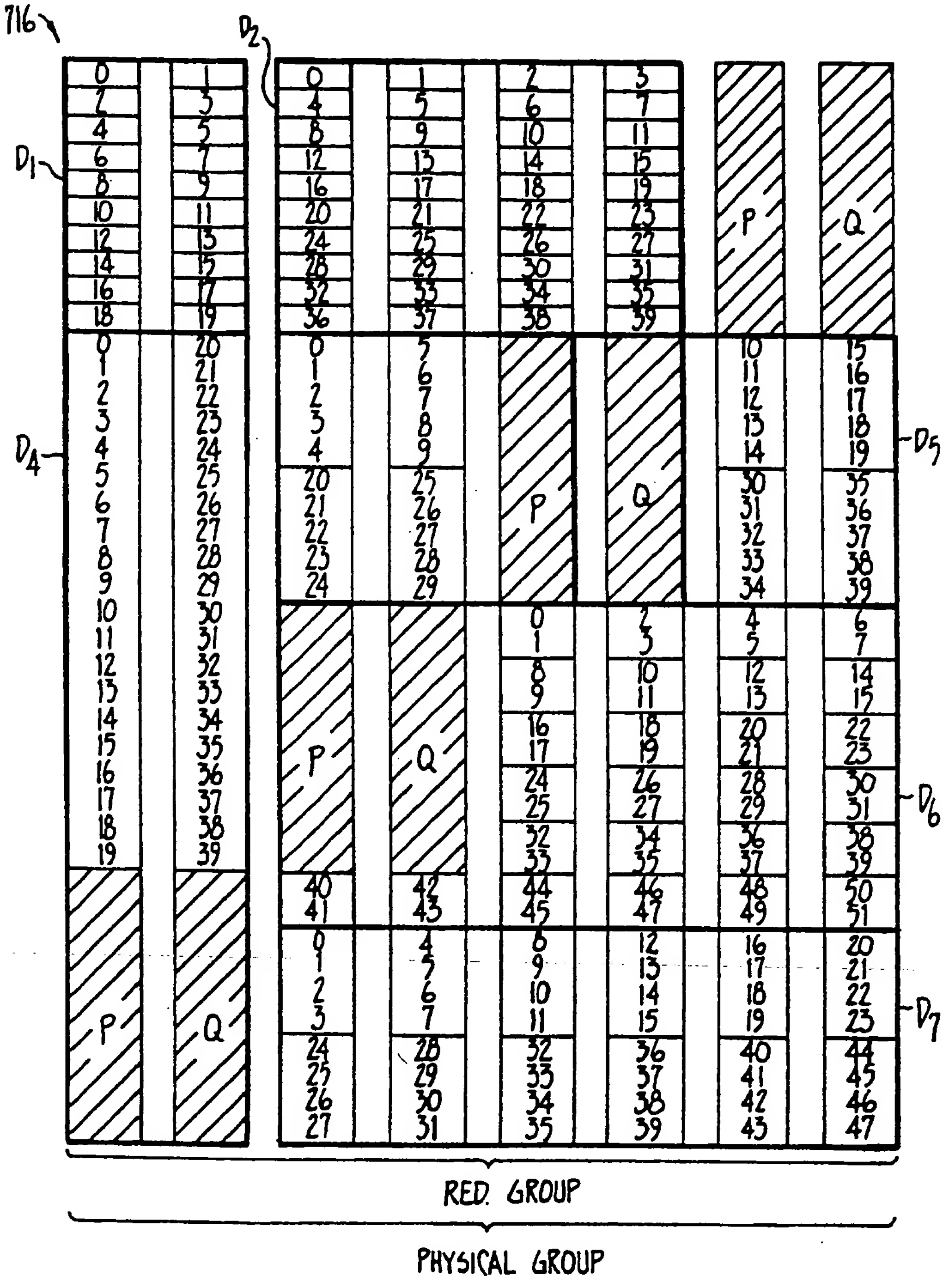


FIG. 23.

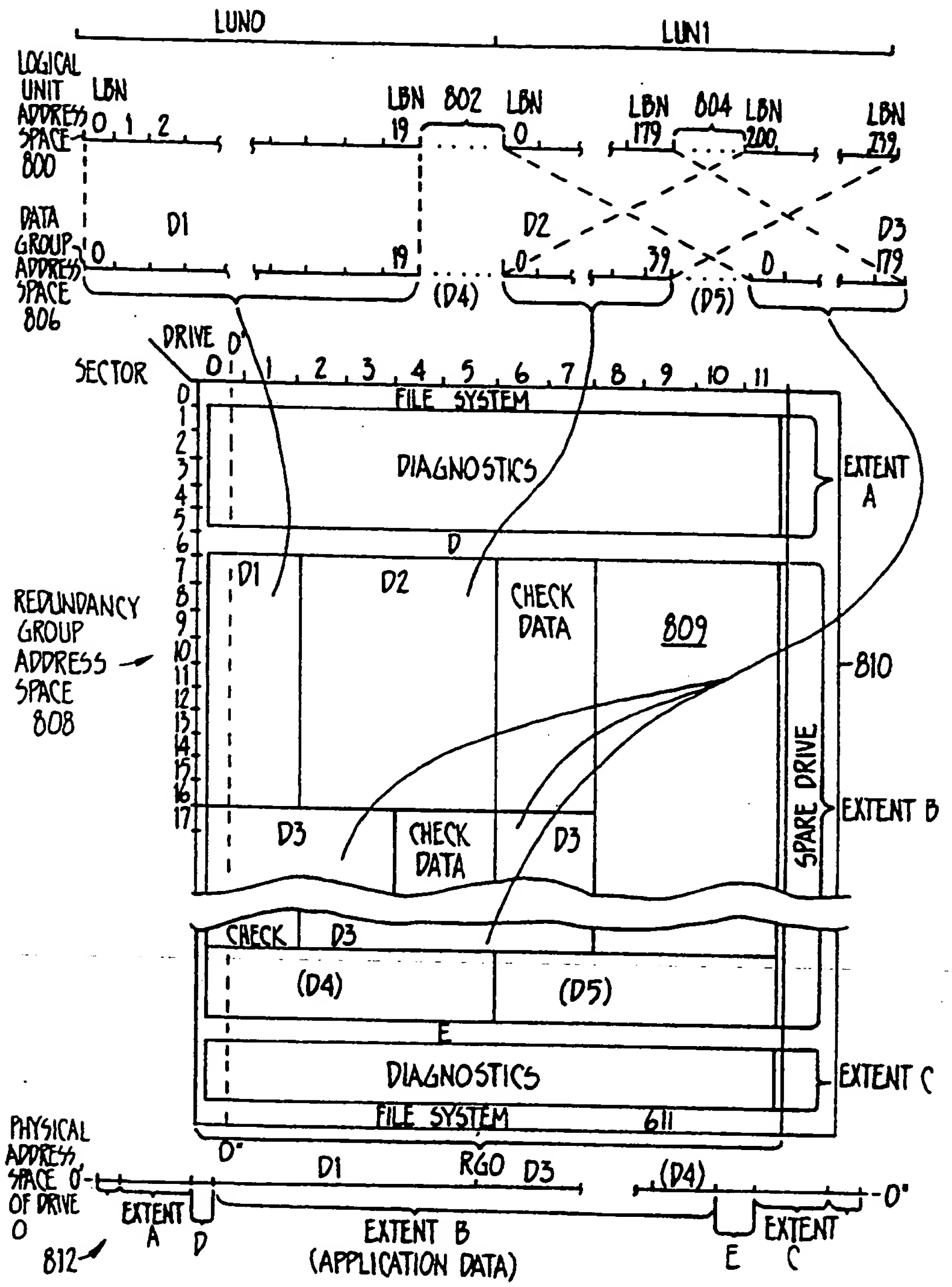

LUN0
LUN1
LOGICAL UNIT ADDRESS SPACE 800
LBN 0, 1, 2
LBN 19
802
LBN 0
LBN 179
804
LBN 200
LBN 239
DATA GROUP ADDRESS SPACE 806
D1
0
19
(D4)
D2
0
39
(D5)
D3
0
179
DRIVE
SECTOR
0' 0 1 2 3 4 5 6 7 8 9 10 11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
FILE SYSTEM
DIAGNOSTICS
EXTENT A
D
D1
D2
CHECK DATA
809
810
REDUNDANCY GROUP ADDRESS SPACE 808
D3
CHECK DATA
D3
SPARE DRIVE
EXTENT B
CHECK
D3
(D4)
(D5)
E
DIAGNOSTICS
EXTENT C
FILE SYSTEM
611
PHYSICAL ADDRESS SPACE 0' OF DRIVE 0
812
0'
D1
RG0
D3
(D4)
-0"
EXTENT A
D
EXTENT B (APPLICATION DATA)
E
EXTENT C

FIG. 24.

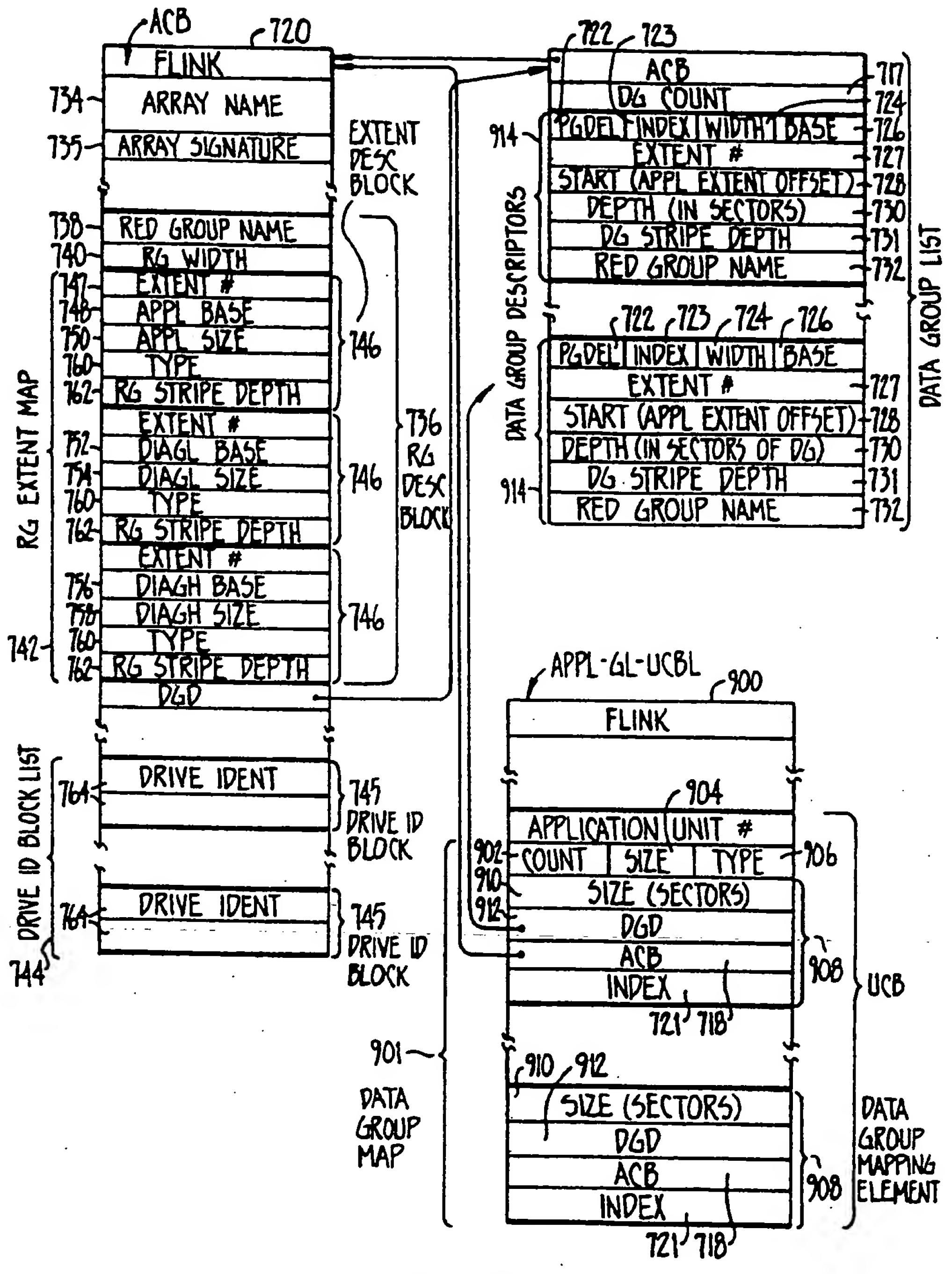

ACB
720
FLINK
734
ARRAY NAME
735
ARRAY SIGNATURE
EXTENT DESC BLOCK
738
RED GROUP NAME
740
RG WIDTH
747
EXTENT #
748
APPL BASE
750
APPL SIZE
760
TYPE
762
RG STRIPE DEPTH
746
EXTENT #
752
DIAGL BASE
754
DIAGL SIZE
TYPE
RG STRIPE DEPTH
736
RG DESC BLOCK
EXTENT #
756
DIAGH BASE
758
DIAGH SIZE
TYPE
RG STRIPE DEPTH
742
RG EXTENT MAP
DGD
764
DRIVE IDENT
745
DRIVE ID BLOCK
DRIVE ID BLOCK LIST
744
722
723
ACB
717
DG COUNT
724
914
PGDEL INDEX WIDTH BASE
726
EXTENT #
727
START (APPL EXTENT OFFSET)
728
DEPTH (IN SECTORS)
730
DG STRIPE DEPTH
731
RED GROUP NAME
732
DATA GROUP DESCRIPTORS
DATA GROUP LIST
DEPTH (IN SECTORS OF DG)
APPL-GL-UCBL
900
FLINK
904
APPLICATION UNIT #
902
COUNT SIZE TYPE
906
910
SIZE (SECTORS)
912
DGD
ACB
INDEX
721
718
908
UCB
901
DATA GROUP MAP
DATA GROUP MAPPING ELEMENT

FIG. 25.

DISK ARRAY SYSTEM

CROSS REFERENCE TO RELATED APPLICATIONS

This is a continuation of application Ser. No. 07/601.482 filed Oct. 22. 1990 which is a continuation-in-part of Ser. Nos. 07/505,622, 07/506,703, and 07/488,749, filed Apr. 6, 1990, Apr. 6, 1990, and Mar. 2, 1990, respectively.

BACKGROUND OF THE INVENTION

The present invention relates generally to memory storage devices. More particularly, the invention is a method and apparatus for interfacing an external computer to a set of storage devices which are typically disk drives.

Magnetic disk drive memories for use with digital computer systems are known. Although many types of disk drives are known, the present invention will be described as using hard disk drives. However, nothing herein should be taken to limit the invention to that particular embodiment.

Many computer systems use a plurality of disk drive memories to store data. A common known architecture for such systems is shown in FIG. 1. Therein, computer **10** is coupled by means of bus **15** to disk array **20**. Disk array **20** is comprised of large buffer **22**, bus **24**, and a plurality of disk drives **30**. The disk drives **30** can be operated in various logical configurations. When a group of drives is operated collectively as a logical device, data stored during a write operation can be spread across one or more members of an array. Disk controllers **35** are connected to buffer **22** by bus **24**. Each controller **35** is assigned a particular disk drive **30**.

Each disk drive within disk drive array **20** is accessed and the data thereon retrieved individually. The disk controller **35** associated with each disk drive **30** controls the input/output operations for the-particular disk drive to which it is coupled. Data placed in buffer **22** is available for transmission to computer **10** over bus **15**. When the computer transmits data to be written on the disks, controllers **35** receive the data for the individual disk drives from bus **24**. In this type of system, disk operations are asynchronous in relationship to each other.

In the case where one of the controllers experiences a failure, the computer must take action to isolate the failed controller and to switch the memory devices formerly under the failed controller's control to a properly functioning other controller. The switching requires the computer to perform a number of operations. First, it must isolate the failed controller. This means that all data flow directed to the failed controller must be redirected to a working controller.

In the system described above, it is necessary for the computer to be involved with rerouting data away from a failed controller. The necessary operations performed by the computer in completing rerouting requires the computer's attention. This places added functions on the computer which may delay other functions which the computer is working on. As a result, the entire system is slowed down.

Another problem associated with disk operations, in particular writing and reading. is an associated probability of error. Procedures and apparatus have been developed which can detect and, in some cases, correct the errors which occur during the reading and writing of the disks. With relation to a generic disk drive, the disk

is divided into a plurality of sectors, each sector having the same, predetermined size. Each sector has a particular header field, which gives the sector a unique address, a header field code which allows for the detection of errors in the header field, a data field of variable length and ECC ("Error Correction Code") codes, which allow for the detection and correction of errors in the data.

When a disk is written to, the disk controller reads the header field and the header field code. If the sector is the desired sector and no header field error is detected, the new data is written into the data field and the new data ECC is written into the ECC field.

Read operations are similar in that initially both the header field and header field error code are read. If no header field errors exist, the data and the data correction codes are read. If no error is detected the data is transmitted to the computer. If errors are detected, the error correction circuitry located within the disk controller tries to correct the error. If this is possible, the corrected data is transmitted. Otherwise, the disk drive's controller signals to the computer or master disk controller that an uncorrectable error has been detected.

In FIG. 2 a known disk drive system which has an associated error correction circuit, external to the individual disk controllers, is shown. This system uses a Reed-Solomon error detection code both to detect and correct errors. Reed-Solomon codes are known and the information required to generate them is described in many references. One such reference is *Practical Error Correction Design for Engineers*, published by Data Systems Technology Corp., Broomfield, Colo. For purposes of this application, it is necessary to know that the Reed-Solomon code generates redundancy terms, herein called P and Q redundancy terms, which terms are used to detect and correct data errors.

In the system shown in FIG. 2, ECC **42** unit is coupled to bus **45**. The bus is individually coupled to a plurality of data disk drives, numbered here **47**, **48**, and **49**, as well as to the P and Q term disk drives, numbered **51** and **53** through Small Computer Standard Interfaces ("SCSIs") **54** through **58**. The American National Standard for Information Processing ("ANSI") has promulgated a standard for SCSI which is described in ANSI document number X3.130-1986.

Bus **45** is additionally coupled to large output buffer **22**. Buffer **22** is in turn coupled to computer **10**. In this system, as blocks of data are read from the individual data disk drives, they are individually and sequentially placed on the bus and simultaneously transmitted both to the large buffer and the ECC unit. The P and Q terms from disk drives **51** and **53** are transmitted to ECC **42** only. The transmission of data and the P and Q terms over bus **45** occurs sequentially. The exact bus width can be any arbitrary size with 8-, 16- and 32-bit wide buses being common.

After a large block of data is assembled in the buffer, the calculations necessary to detect and correct data errors, which use the terms received from the P and Q disk drives, are performed within the ECC unit **42**. If errors are detected, the transfer of data to the computer is interrupted and the incorrect data is corrected, if possible.

During write operations, after a block of data is assembled in buffer **22**, new P and Q terms are generated within ECC unit **42** and written to the P and Q disk drives at the same time that the data in buffer 22 is written to the data disk drives.

Those disk drive systems which utilize known error correction techniques have several shortcomings. In the systems illustrated in FIGS. 1 and 2, data transmission is sequential over a single bus with a relatively slow rate of data transfer. Additionally, as the error correction circuitry must wait until a block of data of predefined size is assembled in the buffer before it can detect and correct errors therein, there is an unavoidable delay while such detection and correction takes place.

As stated, the most common form of data transmission in these systems is serial data transmission. Given that the bus has a fixed width, it takes a fixed and relatively large amount of time to build up data in the buffer for transmission either to the disks or computer. If the large, single buffer fails, all the disk drives coupled thereto become unusable. Therefore, a system which has a plurality of disk drives which can increase the rate of data transfer between the computer and the disk drives and more effectively match the data transfer rate to the computer's maximum efficient operating speed is desirable. The system should also be able to conduct this high rate of data transfer while performing all necessary error detection and correction functions and at the same time provide an acceptable level of performance even when individual disk drives fail.

Another failing of prior art systems is that they do not exploit the full range of data organizations that are possible in a system using a group of disk drive arrays. In other words, a mass storage apparatus made up of a plurality of physical storage devices may be called upon to operate as a logical storage device for two concurrently-running applications having different data storage needs. For example, one application requiring large data transfers (i.e., high bandwidth), and-the other requiring high frequency transfers (i.e., high operation rate). A third application may call upon the apparatus to provide both high bandwidth and high operating rate. Known operating techniques for physical device sets do not provide the capability of dynamically configuring a single set of physical storage devices to provide optimal service in response to such varied needs.

It would therefore be desirable to be able to provide a mass storage apparatus, made up of a plurality of physical storage devices, which could flexibly provide both high bandwidth and high operation rate, as necessary, along with high reliability.

SUMMARY OF THE INVENTION

The present invention provides a set of small, inexpensive disk drives that appears to an external computer as one or more logical disk drives. The disk drives are arranged in sets. Data is broken up and written across the disk drives in a set, with error detection and correction redundancy data being generated in the process and also being written to a redundancy area. Backup disk drives are provided which can be coupled into one or more of the sets. Multiple control systems for the sets are used, with any one set having a primary control system and another control system which acts as its backup, but primarily controls a separate set. The error correction or redundancy data and the error detection data is generated "on the fly" as data is transferred to the disk drives. When data is read from the disk drives, the error detection data is verified to confirm the integrity of the data. Lost data from a particular disk drive can be regenerated with the use of the redundancy data

and a backup drive can be substituted for a failed disk drive.

The present invention provides an arrangement of disk drive controllers, data disk drives and error correction code disk drives, the drives being each individually coupled to a small buffer memory and a circuit for error detection and correction. A first aspect of the present invention is error detection and correction which occurs nearly simultaneously with the transfer of data to and from the disk drives. The multiple buffer memories can then be read from or written to in sequence for transfers on a data bus to the system computer. Additionally, the error correction circuitry can be connected to all of the buffer memory/disk drive data paths through a series of multiplexer circuits called cross-bar ("X-bar") switches. These X-bar switches can be used to decouple failed buffer memories or disk drives from the system.

A number of disk drives are operatively interconnected so as to function at a first logical level as one or more logical redundancy groups. A logical redundancy group is a set of disk drives which share redundancy data. The width, depth and redundancy type (e.g., mirrored data or check data) of each logical redundancy group, and the location of redundant information therein, are independently configurable to meet desired capacity and reliability requirements. At a second logical level, blocks of mass storage data are grouped into one or more logical data groups. A logical redundancy group may be divided into more than one such data group. The width, depth, addressing sequence and arrangement of data blocks in each logical data group are independently configurable to divide the mass data storage apparatus into multiple logical mass storage areas each having potentially different bandwidth and operation rate characteristics.

A third logical level, for interacting with application software of a host computer operating system, is also provided. The application level superimposes logical application units on the data groups to allow data groups, alone or in combination from one or more redundancy groups, to appear to application software as single logical storage units.

As data is written to the drives, the error correction circuit, herein called the Array Correction Circuit ("ACC"), calculates P and Q redundancy terms and stores them on two designated P and Q disk drives through the X-bar switches. In contrast to the discussed prior art, the present invention's ACC can detect and correct errors across an entire set of disk drives simultaneously, hence the use of the term "Array correction Circuit." In the following description, the term ACC will refer only to the circuit which performs the necessary error correction functions. The codes themselves will be referred to as Error Correction Code or "ECC". On subsequent read operations, the ACC may compare the data read with the stored P and Q values to determine if the data is error-free.

The X-bar switches have several internal registers. As data is transmitted to and from the data disk drives, it must go through a X-bar switch. Within the X-bar switch the data can be clocked from one register to the next before going to the buffer or the disk drive. The time it takes to clock the data through the X-bar internal registers is sufficient to allow the ACC to calculate and perform its error correction tasks. During a write operation, this arrangement allows the P and Q values to be generated and written to their designated disk drives at the same time as the data is written to its disk drives, the operations occurring in parallel. In effect the X-bar switches establish a data pipeline of several stages, the plurality of stages effectively providing a time delay circuit.

In one preferred embodiment, two ACC units are provided. Both ACCs can be used simultaneously on two operations that access different disk drives or one can be used if the other fails.

The X-bar switch arrangement also provides flexibility in the data paths. Under control of the system controller, a malfunctioning disk drive can be decoupled from the system by reconfiguring the appropriate X-bar switch or switches and the data that was to be stored on the failed disk can be rerouted to another data disk drive. As the system computer is not involved in the detection or correction of data errors, or in reconfiguring the system in the case of failed drives or buffers, these processes are said to be transparent to the system computer.

In a first embodiment of the present invention, a plurality of X-bar switches are coupled to a plurality of disk drives and buffers, each X-bar switch having at least one data path to each buffer and each disk drive. In operation a failure of any buffer or disk drive may be compensated for by rerouting the data flow through a X-bar switch to any operational drive or buffer. In this embodiment full performance can be maintained when disk drives fail.

In another embodiment of the present invention, two ACC circuits are provided. In certain operating modes, such as when all the disk drives are being written to or read from simultaneously, the two ACC circuits are redundant, each ACC acting as a back-up unit to the other. In other modes, such as when data is written to an individual disk drive, the two ACCs work in parallel, the first ACC performing a given action for a portion of the entire set of drives, while the second ACC performs a given action which is not necessarily the same for a remaining portion of the set.

In yet another embodiment, the ACC performs certain self-monitoring check operations using the P and Q redundancy terms to determine if the ACC itself is functioning properly. If these check operations fail, the ACC will indicate its failure to the control system, and it will not be used in any other operations.

In still another embodiment, the ACC unit is coupled to all the disk drives in the set and data being transmitted to or from the disk drives is simultaneously recovered by the ACC. The ACC performs either error detection or error correction upon the transmitted data in parallel with the data transmitted from the buffers and the disk drives.

The present invention provides a speed advantage over the prior art by maximizing the use of parallel paths to the disk drives. Redundancy and thus fault-tolerance is also provided by the described arrangement of the X-bar switches and ACC units.

Another aspect of the present invention is that it switches control of disk drive sets when a particular controller fails. Switching is performed in a manner transparent to the computer.

The controllers comprise a plurality of first level controllers each connected to the computer. Connected to the other side of the first level controllers is a set of second level controllers. Each first level controller can route data to any one of the second level controllers. Communication buses tie together the second level

controllers and the first level controllers can also communicate between themselves. In a preferred embodiment, the system is configured such that the second level controllers are grouped in pairs. This configuration provides each second level controller with a single associated back-up controller. This configuration provides for efficient rerouting procedures for the flow of data to the disk drives. For ease of understanding, the specification will describe the system configured with pairs of second level controllers. Of course, it should be understood that the second level controllers could be configured in groups of three or other groupings.

A switching function is implemented to connect each of the second level controllers to a group of disk drives. In the case that a second level controller should fail, the computer need not get involved with the rerouting of data to the disk drives. Instead, the first level controllers and the properly working second level controller can handle the failure without the involvement of the computer. This allows the logical configuration of the disk drives to remain constant from the perspective of the computer despite a change in the physical configuration.

There are two levels of severity of failures which can arise in the second level controllers. The first type is a complete failure. In the case of a complete failure, the second level controller stops communicating with the first level controllers and the other second level controller. The first level controllers are informed of the failure by the properly working second level controller or may recognize this failure when trying to route data to the failed second level controller. In either case, the first level controller will switch data paths from the failed second level controller to the properly functioning second level controller. Once this rerouted path has been established, the properly functioning second level controller issues a command to the malfunctioning second level controller to release control of its disk drives. The properly functioning second level controller then assumes control of these disk drive sets.

The second type of failure is a controlled failure where the failed controller can continue to communicate with the rest of the system. The partner second level controller is informed of the malfunction. The properly functioning second level controller then informs the first level controllers to switch data paths to the functioning second level controller. Next, the malfunctioning second level controller releases its control of the disk drives and the functioning second level controller assumes control. Finally, the properly functioning second level controller checks and, if necessary, corrects data written to the drives by the malfunctioning second level controller.

A further aspect of the present invention is a SCSI bus switching function which permits the second level controllers to release and assume control of the disk drives.

For a more complete understanding of the nature and the advantages of the invention, reference should be made to the ensuing detail description taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a prior art disk array system;

FIG. 2 is a block diagram illustrating a prior art disk array system with an error check and correction block;

FIG. 3 is a diagram illustrating a preferred embodiment f the verall system of the present inventi n:

FIG. 3.5 is a diagram showing a pair of disk drive sets 18A and 18B connected to a pair of second level controllers 14A and 14B.

FIG. 4 is a diagram showing a m re detailed illustration of FIG. 3 including the interconnections f the switches and the disk drives within the disk drive sets:

FIG. 5 is a block diagram of the wiring between the controllers and the switches;

FIG. 6 is a block diagram showing the schematic circuitry of the switching function control circuitry shown in FIG. 5;

FIG. 7 is a recovery state transition diagram illustrating the various possible states of a particular second level controller;

FIGS. 8A-8I show the events which take place during the transition between each of the states shown in FIG. 7;

FIG. 9 is a block diagram of one preferred embodiment of the X-bar circuitry;

FIG. 10 is a block diagram of a preferred embodiment of the error check and correction circuitry;

FIG. 11 is a detailed block diagram of the X-bar switches and the ACC shown in FIG. 10;

FIGS. 12*a* and 12*b* show the logic operations necessary to calculate the P and Q error detection terms;

FIGS. 13*a* and 13*b* show how the Reed-Solomon codeword is formed and stored in one embodiment of the present invention;

FIGS. 14*a* and 14*b* show the parity detector and parity generator circuits in the ACC;

FIGS. 15, 16, 17, and 18 show, respectively, the data flow during a Transaction Mode Normal Read, a Transaction Mode Failed Drive Read, a Transaction Mode Read-Modify-Write Read and a Transaction Mode Read-Modify-Write Write;

FIG. 19 is a schematic diagram of a set of disk drives in which check data is distributed among drives of the set according to a known technique;

FIG. 20 is a schematic diagram of a mass storage system suitable for use with the present invention;

FIG. 21 is a schematic diagram of the distribution of data on the surface of a magnetic disk;

FIG. 22 is a schematic diagram of the distribution of data in a first preferred embodiment of a redundancy group according to the present invention;

FIG. 23 is a schematic diagram of the distribution of data in a second, more particularly preferred embodiment of a redundancy group according to the present invention;

FIG. 24 is a diagram showing how the memory space of a device set might be configured in accordance with the principles of the present invention; and

FIG. 25 is a diagram of an exemplary embodiment of data structures for napping between the logical levels of the present invention.

DETAILED DESCRIPTION OF THE DRAWINGS

The preferred embodiments of the present invention comprise a system for mass data storage. In the preferred embodiments described herein, the preferred devices for storing data are hard disk drives, referenced herein as disk drives. Nothing herein should be understood to limit this invention to using disk drives only. Any other device for storing data may be used, including. but not limited to, floppy disks, magnetic tape drives, and optical disks.

Overall System Environment

One preferred embodiment of the present invention operates in the environment sh wn in FIG. 3. In FIG. 3, computer 10 communicates with a group of disk drive sets 18 through controller 11. In a preferred embodiment, controller 11 includes a number of components which permit computer 10 to access each of disk drive sets 18 even when there is a failure in one of the components of controller 11. As shown in FIG. 3, controller 11 includes a pair of two-level devices 13. Within each of the two-level devices 13 is a first level controller 12 and a second level controller 14. A switch 16 which comprises a group of switches permits computer 10 to access disk drive sets 18 through more than one path. In this way, if either of two-level devices 13 experience a failure in one of their components, the path may be re-routed without computer 10 being interrupted.

FIG. 3.5 is a diagram showing a pair of disk drive sets 18A and 18B connected to a pair of second level controllers 14A and 14B. Controllers 14A and 14B each include two interface modules 27 for interfacing second level controllers 14 with a pair of first level controllers 12 (shown in FIG. 3). Interface modules 27 are connected to buffers 330-335 which buffer data to be transmitted to and received from the disk drives. Second level controllers 14 are configured to be primarily responsible for one group of disk drives and secondarily responsible for a second group of disk drives. As shown, second level controller 14A is primarily responsible for disk drives 20A1, 20B1, 20C1, 20D1 and 20E1 and secondarily responsible for disk drives 20A2, 20B2, 20C2, 20D2 and 20E2. A spare drive 20X is shared by both second level controllers and is activated to take over for a disk drive which has failed.

The disk drives are connected to second level controllers 14 through a set of data interfaces 31. These interfaces are set by controllers 14 to configure the disk drives in a particular arrangement. For example, disk drives 20A1, 20B1, 20C1 and 20D1 and 20A2, 20B2, 20C2 and 20D2 may be set to store data for the system while disk drives 20E1 and 20E2 may be set to store error correction codes. If any of the drives fail, drive 20X is set to take its place. Of course, the disk drives of the system can be rearranged and may assume a wide variety of configurations.

FIG. 4 is more detailed diagram showing the interconnection of the components of the input/output system associated with computer 10 for accessing disk drive sets 18. Computer 10 has its input/output ports connected to the first level controllers 12A and 12B. Second level controllers 14A and 14B are shown connected to first level controllers 12A and 12B. The lines between second level controllers 14 and first level controllers 12 represent data buses through which data flows as well as control and status signals. The dashed line between second level controller 14A and second level controller 14B represents a communication line through which the second level controllers communicate with each other.

Second level controllers 14 are each connected to a group of disk drive sets 18A-18F through switches 16A-16F.

Disk drives 20 are arranged in a manner so that each second level controller 14 is primarily responsible for one group f disk drive sets. As shown in FIG. 4, second level controller 14A may be primarily responsible for three of the disk drive sets 18A-18F. Similarly, second level controller 14B may be primarily responsible for the remaining three disk drive sets 18A-18F. Second level controllers 14 are secondarily responsible for the disk drives primarily controlled by the partner second level controller. In the particular arrangement shown in FIG. 4, second level controller 14A may be primarily responsible for the left three disk drive sets 18A, 18B and 18C and secondarily responsible for the right three disk drives sets 18D, 18E and 18F. Second level controller 14B is primarily responsible for the right three disk drive sets 18D, 18E and 18F and secondarily responsible for the left three disk sets 18A, 18B and 18C.

Each second level controller 14 contains a second level controller recovery system (CRS) 22. CRS 22 is a portion of software code which manages the communication between second level controllers 14 and first level controllers 12. CRS 22 is typically implemented as a state machine which is in the form of microcode or sequencing logic for moving second level controller 14 from state to state (described below). State changes are triggered as different events occur and messages are sent between the various components of the system.

An ECC block 15 is also included in each second level controller 14. ECC block 15 contains circuitry for checking and correcting errors in data which occur as the data is passed between various components of the system. This circuitry is described in more detail below.

FIG. 5 is a block diagram showing a more detailed illustration of the interconnections between second level controllers 14A and 14B and the disk drives. For simplicity, only a single disk drive port is shown. More disk drive ports are included in the system as shown in FIGS. 3 and 4.

Second level controller 14A has a primary control/-sense line 50A for controlling its primary set of disk drives. An alternate control/sense line 52A controls an alternate set of disk drives. Of course, second level controller 14B has a corresponding set of control/sense lines. Data buses 54A (second level controller 14A) and 54B (second level controller 14B) carry the data to and from disk drives 20. These data buses are typically in the form of a SCSI bus.

A set of switches 16A-16F are used to grant control of the disk drives to a particular second level controller. For example, in FIG. 4, second level controller 14A has primary responsibility for disk drives 20A-20C and alternate control of disk drives 20D-20F. Second level controller 14B has primary control of disk drives 20D-20F and alternate control of disk drives 20A-20C. By changing the signals on control/sense lines 50 and 52, primary and secondary control can be altered.

FIG. 6 is a more detailed illustration of one of the switches 16A-16F. A pair of pulse shapers 60A and 60B receive the signals from the corresponding control/-sense lines 50A and 52B shown in FIG. 5. Pulse shapers 60 clean up the signals which may have lost clarity as they were transmitted over the lines. Pulse shapers of this type are well% known in the art. The clarified signals from pulse shapers 60 are then fed to the set and reset pins of R/S latch 62. The Q and $\underline{Q}$ outputs of latch 62 are sent to the enable lines of a pair of driver/receivers 64A and 64B. Driver/receivers 64A and 64B are connected between the disk drives and second level controllers 14A and 14B. Depending upon whether primary control/sense line 52B or alternate control/-sense line 50A is active, the appropriate second level controller will be in control at a particular time.

FIG. 7 is a state transition diagram showing the relationships between the various states of CRS 22 (FIG. 3) of a particular second level controller 14. Each second level controller 14 must be in only one state at any particular point in time. Initially, assuming that the system is functioning properly and each second level controller 14 is primarily responsible for half of the disk drive sets 18 and secondarily responsible for half of the disk drive sets 18, second level controller 14 is in a PRIMARY STATE 26. While in PRIMARY STATE 26, two major events may happen to move a second level controller 14 from PRIMARY STATE 26 to another state. The first event, is the failure of the particular second level controller 14. If there is a failure, second level controller 14 shifts from PRIMARY STATE 26 to a NONE STATE 28. In the process of doing so, it will pass through RUN-DOWN-PRIMARIES-TO-NONE STATE 30.

There are two types of failures which are possible in second level controller 14. The first type of failure is a controlled failure. Further, there are two types of controlled failures.

The first type of controlled failure is a directed controlled failure. This is not actually a failure but instead an instruction input from an outside source instructing a particular second level controller to shut down. This instruction may be received in second level controller 14 from one of the following sources: An operator, through computer 10; a console 19 through a port 24 (e.g. RS-232) on the first level controller; a diagnostic console 21 through a port 23 (e.g. RS-232) on the second level controller; or by software initiated during predictive maintenance. Typically, such an instruction is issued in the case where diagnostic testing of a second level controller is to be conducted. In a directed controlled failure, the second level controller finishes up any instructions it is currently involved with and refuses to accept any further instructions. The second level controller effects a "graceful" shut down by sending out messages to the partner second level controller that it will be shutting down.

The second type of controlled failure is referred to as a moderate failure. In this case, the second level controller recognizes that it has a problem and can no longer function properly to provide services to the system. For example, the memory or drives associated with that second level controller may have malfunctioned. Therefore, even if the second level controller is properly functioning, it cannot adequately provide services to the system. It aborts any current instructions, refuses to accept any new instructions and sends a message to the partner second level controller that it is shutting down. In both controlled failures, the malfunctioning second level controller releases the set of disk drives over which it has control. These drives are then taken over by the partner second level controller.

The second type of failure is a complete failure. In a complete failure, the second level controller becomes inoperable and cannot send messages or "clean-up" its currently pending instructions by aborting them. In other words, the second level controller has lost its ability to serve the system. It is up to one of the first level controllers or the partner second level controller to recognize the problem. The partner second level controller then takes control of the drives controlled by the malfunctioning second level controller. The routing through the malfunctioning second level controller is switched over to the partner second level controller.

In all of the above failures, the switching takes place without interruption to the operation of the computer. Second level controllers 14 and first level controllers 12 handle the rerouting independently by communicating the failure among themselves.

Assuming there was a failure in second level controller 14A, second level controller 14A moves from PRIMARY STATE 26 through a transition RUN-DOWN-PRIMARIES-TO-NONE STATE 30 to NONE STATE 28. At the same time, properly functioning second level controller 14B moves from PRIMARY STATE 26 to BOTH STATE 32. The basis for the change in state of each of second level controllers 14A and 14B is the failure of second level controller 14A. When a second level controller fails, it is important to switch disk drive control away from the failed second level controller. This permits computer 10 to continue to access disk drives which were formerly controlled by a particular second level controller which has failed. In the current example (FIG. 4), disk drive sets 18A-18C are switched by switching functions 16A-16C so that they are controlled by second level controller 14B. Therefore, second level controller 14B is in BOTH STATE 32 indicating that it has control of the disk drive sets 18 for both second level controllers. Second level controller 14A now controls none of the disk drives and is in NONE STATE 28. The transition state 30 determines which of several possible transition paths is used.

If second level controller 14A is in NONE STATE 28 and second level controller 14B is in BOTH STATE 32 there are a number of options for transferring control of disk drive sets 18A-18F once second level controller 14A has been repaired. First, second level controller 14A and second level controller 14B could each be shifted back to PRIMARY STATE 26. This is accomplished for drive sets 18A-18C by second level controller 14A moving from NONE STATE 28 directly to PRIMARY STATE 26 along the preempt p line. Preempt p simply stands for "preempt primary" which means that second level controller 14A preempts its primary drives or takes control of them from second level controller 14B. At the same time, second level controller 14B moves from BOTH STATE 32 through a transition RUN-DOWN-SECONDARIES-TO-PRIMARIES STATE 34 and then to PRIMARY STATE 26.

A second alternative is for second level controller 14A to move from NONE STATE 28 to SECONDARY STATE 36. Once in SECONDARY STATE 36, second level controller 14A is in control of its secondary disk drive sets 18D-18F. Second level controller 14B concurrently moves from BOTH STATE 32 through RUN-DOWN-PRIMARIES-TO-SECONDARIES STATE 38 and on to SECONDARY STATE 36. When both second level controllers are in SECONDARY STATE 36, they are in control of their secondary disk drive sets. Second level controller 14A controls disk drive sets 18D-18F and second level controller 14B controls disk drive sets 18A-18C.

From SECONDARY STATE 36, a failing second level controller 14 may move through RUN-DOWN-SECONDARIES-TO-NONE STATE 40 to NONE STATE 28. If this occurs, the properly functioning partner second level controller 14 moves from SECONDARY STATE 36 to BOTH STATE 32 so that

computer 10 could access any one of disk drive sets 18. As in the previous example, if second level controller 14A were t fail it moves from SECONDARY STATE 36 through RUN-DOWN-SECONDARIES-TO-NONE STATE 40 and into NONE STATE 28. At the same time, properly functioning second level controller 14B moves from SECONDARY STATE 36 along the preempt b/p line into BOTH STATE 32. Preempt b/p stands for "preempt both/primaries." In other words, all of the disk drives are preempted by the properly functioning second level controller.

If, for all sets 18, second level controller 14A is in NONE STATE 28 and second level controller 14B is in BOTH STATE 32, it is possible for second level controller 14A to take control of all sets 18 of disk drives. This is desirable if second level controller 14A were repaired and second level controller 14B failed. Second level controller 14A moves from NONE STATE 28 along the preempt b line to BOTH STATE 32. At the same time, second level controller 14B moves from BOTH STATE 32 through RUN-DOWN-BOTH-TO-NONE STATE 42 and into NONE STATE 28. At this point, second level controller 14A controls all disk drives while second level controller 14B controls none of the disk drives.

Various failures may trigger the movement of second level controllers 14 between states. Between states a number of events take place. Each of these events is described in FIGS. 8A-8I. In FIG. 8A, second level controller 14 is in PRIMARY STATE 26. There are three different events which can take place while second level controller 14 is in PRIMARY STATE 26. The first event is for a preempt message 100 to be received from the partner second level controller. At this point, the second level controller receiving such a message will take the secondary path, represented by block 102, and end up at BOTH STATE 32. The second path which may be taken is triggered by receipt of a message 104 from CRS 22 of the other second level controller. This may be some sort of communication which results in the second level controller remaining in PRIMARY STATE 26. It will report and return messages 106 to the other second level controller. The final path which may be taken results in second level controller ending up in RUN-DOWN-PRIMARIES-TO-NONE STATE 30. This path is triggered upon receipt of a message 108 to release both sets of drives or the primary disk drives. A timer is then set in block 110 and upon time out a message 112 is sent to the other second level controller to take control of the primary set of disk drives. Once in RUN-DOWN-PRIMARIES-TO-NONE STATE 30, second level controller 14 will eventually end up in NONE STATE 28.

FIG. 8B illustrates various paths from RUN-DOWN-PRIMARIES-TO-NONE STATE 30 to NONE STATE 28. Three possible events may take place. First, a message 114 may be received from another second level controller providing communication information. In this case, second level controller 14 reports back messages 116 and remains in RUN-DOWN-PRIMARIES-TO-NONE STATE 30. The second event which may occur is for the timer, set during transition from PRIMARY STATE 26 to RUN-DOWN-PRIMARIES-TO-NONE STATE 30 to time out 118. If this happens, second level controller 14 realizes that message 112 (FIG. 8A) didn't get properly sent and that there has been a complete failure. It releases control of both its primaries and secondary disk drives 122. It then ends up in NONE STATE 28. The third event which may occur while in RUN-DOWN-PRIMARIES-TO-NONE STATE 30 is for a response to be received 124 from message 112 (FIG. 8A) sent out while second level controller moved from PRIMARY STATE 26 to RUN-DOWN-PRIMARIES-TO-NONE STATE 30. This response indicates that the message was properly received. Second level controller 14 then releases its primary drives 126 and ends up in NONE STATE 28.

FIG. 8C covers the state transition between NONE STATE 28 and one of either BOTH STATE 32, PRIMARY STATE 26, or SECONDARY STATE 36. When in NONE STATE 28, second level controller 14 can only receive messages. First, it may receive a message 128 instructing it to preempt both its primary and alternative sets of disk drives. It performs this function 130 and ends up in BOTH STATE 32. A second possibility is for it to receive a preempt message 132 instructing it to preempt its primary set of drives. It performs this instruction and ends up in PRIMARY STATE 26. A third alternative is the receipt of a preempt message 136 instructing second level controller 14 to preempt its secondary drives. Upon performance of this instruction 138 it ends up in SECONDARY STATE 36. Finally, while in NONE STATE 28 second level controller 14 may receive communication messages 140 from its partner second level controller. It reports back 142 to the other second level controller and remains in NONE STATE 28.

FIG. 8D illustrates the movement of second level controller 14 from SECONDARY STATE 36 to BOTH STATE 32 or RUN-DOWN-SECONDARIES-TO-NONE STATE 40. While in SECONDARY STATE 36, any one of three messages may be received by second level controller 14. A first possibility is for a preempt both or primary message 144 to be received. At this point, second level controller 14 takes control of its primary drives 146 and ends up in BOTH STATE 32. A second possibility is for communication messages 148 to be received from the partner controller. This results in second level controller 14 reporting back 150 and remaining in its present SECONDARY STATE 36. Finally, a release both or secondary message 152 may be received. Second level controller 14 sets a timer 154 upon receipt of this message. It then sends out a message 156 indicating it is now in RUN-DOWN-SECONDARIES-TO-NONE STATE 40.

FIG. 8E shows the transition of second level controller 14 from RUN-DOWN-SECONDARIES-TO-NONE STATE 40 to NONE STATE 28. Three different messages may be received during RUN-DOWN-SECONDARIES-TO-NONE STATE 40. First, messages 158 from the partner second level controller may be received. Second level controller 14 then reports back (160) to its partner and remains in RUN-DOWN-SECONDARIES-TO-NONE STATE 40. A second possibility is for the timer, set between SECONDARY STATE 36 and the present state, to time out (162). This indicates that message 156 (FIG. 8D) was not properly sent out and received by the partner second level controller and that there has been a complete failure to second level controller 14. Second level controller 14 then reports out (164) that it will release both of its sets of disk drives 166. This results in it moving to NONE STATE 28. Finally, second level controller 14 may receive a response 168 to its message 156 (FIG. 8D) sent after setting the timer between SECONDARY STATE 36 and RUN-DOWN-SECONDARIES-TO-NONE STATE 40. Upon receiving this response, it releases its secondary drives and ends up in NONE STATE 28.

FIG. 8F illustrates the various paths from BOTH STATE 32 to any one of RUN-DOWN-PRIMARIES-TO-SECONDARIES STATE 38, RUN-DOWN-SECONDARIES-TO-PRIMARIES STATE 34 or RUN-DOWN-BOTH-TO-NONE STATE 42. A first possible message which may be received during BOTH STATE 32 is a release primary message 172. This will cause second level controller 14 to set a timer 174, send a message 176 indicating it is running down primaries, and wait in RUN-DOWN-PRIMARIES-TO-SECONDARIES STATE 38. A second message which may be received is a release secondaries message 180. Upon receiving release secondaries message 180, second level controller 14 sets a timer 182 and sends a message 184 indicating it has moved into RUN-DOWN-SECONDARIES-TO-PRIMARIES STATE 34. A third possibility for second level controller 14 is to receive communication messages 186 from its partner second level controller. It will report back (188) and remain in BOTH STATE 32. Finally, second level controller 14 may receive an instruction 190 telling it to release both primary and secondary sets of drives. At this point it sets the timer 192 and sends out a message 194 that it has released both primary and secondary drive sets. It will then remain in the RUN-DOWN-BOTH-TO-NONE STATE 42 until it receives further instructions from the other second level controller.

FIG. 8G shows the various paths by which second level controller 14 moves from RUN-DOWN-PRIMARIES-TO-SECONDARIES STATE 38 to one of either NONE STATE 28 or SECONDARY STATE 36. The first possibility is that second level controller 14 receives messages 196 from the other second level controller. It then reports back (198) and remains in RUN-DOWN-PRIMARIES-TO-SECONDARIES STATE 38. A second possibility is that the timer (174), set between BOTH STATE 32 and RUN-DOWN-PRIMARIES-TO-SECONDARIES STATE 38 times out (200). At this point, second level controller 14 realizes that message 176 (FIG. 8F) was not properly sent. A complete failure has occurred. The second level controller reports (202) that it has released both sets of disk drives, and releases both sets (204). Second level controller 14 then enters NONE STATE 28. Finally, a run down path response message 206 is received acknowledging receipt of message 176 (FIG. 8F) sent between BOTH STATE 32 and RUN-DOWN-PRIMARIES-TO-SECONDARIES STATE 38. Second level controller 14 releases its primary drives 208 and enters SECONDARY STATE 36.

FIG. 8H shows the possible paths down which second level controller 14 moves between RUN-DOWN-SECONDARIES-TO-PRIMARIES STATE 34 and one of either NONE STATE 28 or PRIMARY STATE 26. A first possibility is that second level controller 14 receives a message 210 from the other second level controller. It then reports back (212) and remains in RUN-DOWN-SECONDARIES-TO-PRIMARIES STATE 34. A second possibility is that the timer (182), set between BOTH STATE 32 and RUN-DOWN-SECONDARIES-TO PRIMARY-STATE 34 times out (214). If this occurs, second level controller 14 realizes that message 184 (FIG. 8F) was not properly sent. A complete failure has occurred. Second level controller then sends a message 216 indicating that it has released its drives and then it releases both primary and secondary disk drive sets (218) which it controls. Second level controller then moves into NONE STATE 28. Finally, a third possibility is that second level controller 14 receives a response 220 to message 184 (FIG. 8F) sent between BOTH STATE 32 and RUN-DOWN-SECONDARIES-TO-PRIMARIES-STATE 34. It will then release (222) its secondary drives and enter PRIMARY STATE 26.

FIG. 8I shows the possible paths illustrating the transition of second level controller between RUN-DOWN-BOTH-TO-NONE STATE 42 and NONE STATE 28. Three possible events may take place. First, a message 230 may be received from the other second level controller providing communication information. In this case, second level controller 14 reports back messages 232 and remains in RUN-DOWN-BOTH-TO-NONE STATE 42. The second event which may occur is for the timer (192), set during transition from BOTH STATE 32 to RUN-DOWN-BOTH-TO-NONE STATE 42, to time out (234). If this happens, second level controller 14 realizes that message 194 (FIG. 8F) sent during BOTH STATE 32 didn't get properly sent and that there has been a complete failure. It releases control of both its primaries and secondary disk drives (238). It then ends up in NONE STATE 28. The third event which may occur while in RUN-DOWN-BOTH-TO-NONE STATE 42 is for a response to be received (240) from message 194 (FIG. 8F) sent out while second level controller moved from BOTH STATE 32 to RUN-DOWN-BOTH-TO-NONE STATE 42. This response indicates that the message was properly received. Second level controller 14 then releases both sets of drives (242) and ends up in NONE STATE 28.

Rerouting Data Paths Between Buffers and Disk Drives

FIG. 9 illustrates a first preferred embodiment of circuitry for rerouting data paths between buffers and disk drives 20. In FIG. 9, X-bar switches 310 through 315 are coupled to a bus 309 communicating with the second level controller engine (see FIGS. 3 and 4). In turn, each X-bar switch is coupled by a bus to disk drives 20A1 through 20A6 and to each buffer 330 through 336. Bus 350 couples each buffer to a first level controller which are coupled to a computer such as computer 10 (FIGS. 3 and 4). In this embodiment, although only six disk drives are illustrated, any arbitrary number could be used, as long as the illustrated architecture is preserved by increasing the number of X-bar switches and output buffers in a like manner and maintaining the interconnected bus structures illustrated in FIG. 9.

In operation, the second level controller will load various registers (not illustrated herein) which configure the X-bar switches to communicate with particular buffers and particular disk drives. The particular configuration can be changed at any time while the system is operating. Data flow is bi-directional over all the buses. By configuring the X-bar switches, data flowing from any given buffer may be sent to any given disk drive or vice versa. Failure of any particular system element does not result in any significant performance degradation, as data flow can be routed around the failed element by reconfiguring the registers for the X-bar switch. In a preferred mode of operation, data may be transferred from or to a particular disk drive in parallel with other data transfers occurring in parallel on every other disk drive. This mode of operation allows for a

very high input/output rate of data as well as a high data thr ughput.

T illustrate this embodiment's mode f operation, the following example is offered. Referring to FIG. 9, assume that all data flow is initially direct, meaning, for example, that data in buffer 330 flows directly through X-bar switch 110 to disk drive 20A1. Were buffer 330 t fail, the registers of X-bar switch 310 could be reconfigured, enabling X-bar switch 310 to read data from buffer 335 and direct that data to disk drive 20A1. Similar failures in other buffers and in the disk drives could be compensated for in the same manner.

Generation of Redundancy Terms and Error Detection on Parallel Data

FIG. 10 illustrates a second preferred embodiment of the present invention. This second embodiment incorporates Array Correction Circuits ("ACCs") to provide error detection and correction capabilities within the same general architecture as illustrated for the first preferred embodiment shown in FIG. 9. To ease the understanding of this embodiment, the full details of the internal structure of both the X-bar switches (310 through 315) and the ACC circuits 360 and 370 are not shown in FIG. 10. FIGS. 11 and 12 illustrate the internal structure of these devices and will be referenced and discussed in turn. Additionally, bus LBE as illustrated in FIG. 10 does not actually couple the second level controller (FIGS. 3 and 4) directly to the X-bar switches, the ACCs, and the DSI units. Instead, the second level controller communicates with various sets of registers assigned to the X-bar switches, the ACCs and the DSI units. These registers are loaded by the second level controller with the configuration data which establishes the operating modes of the aforementioned components. As such registers are known, and their operation incidental to the present invention, they are not illustrated or discussed further herein.

The embodiment shown in FIG. 10 shows data disk drives 20A1 through 20A4 and P and Q redundancy term drives 20A5 and 20A6. A preferred embodiment of the present invention utilizes 13 disk drives: ten for data, two for P and Q redundancy terms, and one spare or backup drive. It will be understood that the exact number of drives, and their exact utilization may vary without in any way changing the present invention. Each disk drive is coupled by a bi-directional bus (Small Computer Standard Interface) to units 340 through 345, herein labelled DSI. The DSI units perform some error detecting functions as well as buffering data flow into and out of the disk drives.

Each DSI unit is in turn coupled by a bi-directional bus means to an X-bar switch, the X-bar switches herein numbered 310 through 315. The X-bar switches are coupled in turn to word assemblers 350 through 355 by means of a bi-directional bus. The bus width in this embodiment is 9 bits, 8 for data, 1 for a parity bit. The word assemblers assemble 36-bit (32 data and 4 parity) words for transmission to buffers 330 through 335 over bi-directional buses having a 36-bit width. When data flows from the output buffers to the X-bar switches, the word assemblers decompose the 36-bit words into the 9-bits of data and parity.

The X-bar switches are also coupled to ACC units 348 and 349. The interconnection between the X-bar switches and the ACCs is shown in more detail in FIG. 11. Each X-bar switch can send to both or either ACC the 8 bits of data and 1 parity bit that the X-bar switch receives from either the DSI units or the word assemblers. In turn, the X-bar switches can receive 9 bits of the P and redundancy terms calculated by the ACCs ver lines E_1 and E_2. As shown, the ACCs can direct the P and Q redundancy terms to any X-bar switch, not being limited to the disk drives labelled P and Q. Depending on the configuration commanded by the second level controller, ACCs **348** and **349** can be mutually redundant, in which case the failure of one or the other ACC does not affect the system's ability to detect or correct errors, or each ACC can detect and correct errors on a portion of the total set of disk drives. When operating in this second manner, certain specific types of operations which write data to individual disk drives are expedited, as each ACC can write to a separate individual disk drive. The specific disk drives that the individual ACCs monitor can be reconfigured at any time by the second level controller.

The illustrated connections of the ACCs and the X-bar switches also allows data to be switched from any X-bar switch to any ACC once the second level controller configures the related registers. This flexibility allows data to be routed away from any failed disk drive or buffer.

FIG. **11** shows important internal details of the ACCs and the X-bar switches. X-bar switch **310** is composed of two mirror-image sections. These sections comprise, respectively, 9-bit tristate registers **370**/**380**, lo multiplexers **372**/**382**, first 9-bit registers **374**/**384**, second 9-bit registers **376**/**386**, and input/output interfaces **379**/**389**. In operation, data can flow either from the word assembler to the DSI unit or vice versa.

Although many pathways through the X-bar switch are possible, as shown by FIG. **11**, two aspects of these pathways are of particular importance. First, in order to allow the ACC sufficient time to calculate P and Q redundancy terms or to detect and correct errors, a data pathway of several registers can be used, the data requiring one clock cycle to move from one register to the next. By clocking the data through several registers, a delay of sufficient length can be achieved. For example, assuming a data flow from the word assembler unit to a disk drive, 9 bits are clocked into 9-bit register **374** and tri-state register **370** on the first clock pulse. On the next clock pulse, the data moves to 9-bit register **386** and through redundancy circuit **302** in the ACC **348** to P/Q registers **304** and **306**. The next clock pulses move the data to the DSI unit.

The second important aspect of the internal pathways relates to the two tristate registers. The tri-state registers are not allowed to be active simultaneously. In other words, if either tristate register **370** or **380** is enabled, its counterpart is disabled. This controls data transmission from the X-switch to the ACC. The data may flow only from the DSI unit to the ACC or from the word assembler to the ACC, but not from both to the ACC simultaneously. In the opposite direction, data may flow from the ACC to the word assembler and the DSI simultaneously.

ACC unit **348** comprises a redundancy circuit **302**, wherein P and Q redundancy terms are generated, P and Q registers **304** and **306**, wherein the P and Q redundancy terms are stored temporarily, regenerator and corrector circuit **308**, wherein the data from or to a failed disk drive or buffer can be regenerated or corrected, and output to interfaces **390**, **391**, **392** and **393**.

Redundancy Generation and Error Checking Equations

The main functional components of the second preferred embodiment and their physical connections to one another have now been described. The various preferred modes of operation will now be described. In order to understand these functional modes, some understanding of the error detection and correction method used by the present invention will be necessary.

Various error detection and correction codes are known and used in the computer industry. Error-Control Coding and Applications, D. Wiggert, The MITRE Corp., describes various such codes and their calculation. The present invention in this second preferred embodiment is implemented using a Reed-Solomon error detection and correction code. Nothing herein should be taken to limit the present invention to using only a Reed-Solomon code. If other codes were used, various modifications to the ACCs would be necessary, but these modifications would in no way change the essential features of this invention.

Reed-Solomon codes are generated by means of a field generator polynomial, the one used in this embodiment being X^4+X+1. The code generator polynomial needed for this Reed-Solomon code is $(X+a^0)\cdot(X+a^1)=X^2+a^4X+a^1$. The generation and use of these codes to detect and correct errors is known.

The actual implementation of the Reed-Solomon code in the present invention requires the generation of various terms and syndromes. For purposes of clarity, these terms are generally referred to herein as the P and Q redundancy terms. The equations which generate the P and Q redundancy terms are:

$$P=d_{n-1}+d_{n-2}+\ldots+d_1+d_0$$

and

$$Q=d_{n-1}\cdot a_{n-1}+d_{n-2}\cdot a_{n-2}+\ldots+d_1\cdot a_1+d_0\cdot a_0.$$

The P redundancy term is essentially the simple parity of all the data bytes enabled in the given calculation. The Q logic calculates the Q redundancy for all data bytes that are enabled. For Q redundancy, input data must first be multiplied by a constant "a" before it is summed. The logic operations necessary to produce the P and Q redundancy terms are shown in FIGS. **12***a* and **12***b*. All operations denoted by $\oplus$ are exclusive-OR ("XOR") operations. Essentially, the final P term is the sum of all P_i terms. The Q term is derived by multiplying all Q_i terms by a constant and then XORing the results. These calculations occur in redundancy circuit **302** in ACC **260** (FIG. **11**). The second preferred embodiment, using its implementation of the Reed-Solomon code, is able to correct the data on up to two failed disk drives.

The correction of data requires the generation of additional terms S_0 and S_1 within the ACC. Assuming that the P and Q redundancy terms have already been calculated for a group of data bytes, the syndrome equations

$$S_0=d_{n-1}+d_{n-2}+\ldots+d_1+d_0+P$$

$$S_1=(d_{n-1}\cdot a_{n-1})+(d_{n-2}\cdot a_{n-2})+\ldots+(d_1\cdot a_1)+(d_0\cdot a_0)+Q$$

are used to calculate S_0 and S_1. For S_0 an ACC register enables the necessary data bytes and the P redundancy t be used in the calculation. For S_1, the necessary input data must first be multiplied by a_i before being summed with the Q redundancy information.

As stated, an ACC can correct the data on up to two failed disk drives in this embodiment. The failed disk drive register (not illustrated) in the relevant ACC will be loaded with the address of the failed disk or disks by the second level controller. A constant circuit within the ACC will use the drive location information to calculate two constants k_0 and k_1 as indicated in Table 1 below, where i represents the address of the first failed disk drive, j is the address of the second failed disk drive, and a is a constant. The columns labelled Failed Drives indicate which drives have failed. Column k_0 and k_1 indicate how those constants are calculated given the failure of the drives noted in the Failed Drives columns.

TABLE 1

Failed Drives		k_0	k_1
P	—	0	1
Q	—	1	0
i	—	0	1/ai
i	P	0	1/ai
Q	i	0	0
i	j	aj/ai+aj	1/ai+aj
P	Q	0	0

The error correction circuits use the syndrome information S_0 and S_1, as well as the two constants k_0 and k_1 to generate the data contained on the failed disk drives. The error correction equations are as follows:

$$F_1 = S_0 \cdot k_0 + S_1 \cdot k_1$$

$$F_2 = S_0 + E_1$$

F_1 is the replacement data for the first failed disk drive. F_2 is the replacement data for the second failed disk drive. The equations which generate the P and Q redundancy terms are realized in combinatorial logic, as is partially shown in FIGS. *12a* and *12b*. This has the advantage of allowing the redundancy terms to be generated and written to the disk drives at the same time that the data is written to the drives. This mode of operation will be discussed later.

Operational Modes

Having described the aspects of the Reed-Solomon code implementation necessary to understand the present invention, the operational modes of the present invention will now be discussed.

The second preferred embodiment of the present invention operates primarily in one of two classes of operations. These are parallel data storage operations and transaction processing operation. These two classes of operations will now be discussed with reference to the figures, particularly FIGS. 10, 13 and 14 and Tables 2 through 7.

Although FIG. 10 only shows four data drives and the P and Q redundancy term drives, a preferred embodiment uses a set of 13 disk drives, 10 for data, 2 for the P and Q terms, and a spare. Although nothing herein should be construed to limit this discussion to that specific embodiment, parallel processing operations will be described with relation to that environment.

Parallel Processing Operations

In parallel processing operations, all the drives are considered to comprise a single large set. Each of the disk drives will either receive or transmit 9 bits of data simultaneously. The result of this is that the 9-bits of data appearing in the DSI units of all the drives simultaneously are treated as one large codeword. This result is shown in FIG. *13a*. Codeword 400 comprises 9 bits of data from or for disk drive d_{n-1}, 9 bits of data from or for disk drive d_{n-2}, and so on, with the P and Q disk drives receiving or transmitting the P and Q redundancy term. In a parallel write operation, all the disk drives in the set, except for the spare disk drive, will receive a byte of data (or a redundancy term whose length is equal to the data byte) simultaneously. As shown, the same sector in all the disk drives will receive a part of codeword 400. For example, in the illustration, sector 1 of disk drive $n-1$ will receive a byte of data designated d_{n-1} from codeword 400, sector 1 of disk drive $n-2$ will receive a byte of data designated d_{n-2} from codeword 400 and so on.

In the actual implementation of this preferred embodiment, the codewords are "striped" across the various disk drives. This means that for-each successive codeword, different disk drives receive the P and Q redundancy terms. In other words, drive d_{n-1} is treated as drive d_{n-2} for the second codeword and so on, until what was originally drive d_{n-1} receives a Q redundancy term. Thus, the redundancy terms "stripe" through the disk drives.

Pairs of P and Q Terms for Nibbles

Calculating the P and Q redundancy terms using 8-bit symbols would require a great deal of hardware. To reduce this hardware overhead, the calculations are performed using 4-bit bytes or nibbles. This hardware implementation does not change the invention conceptually, but does result in the disk drives receiving two 4-bit data nibbles combined to make one 8-bit byte. In FIG. *13b*, codeword 450, as well as the illustrated sectors A of the disk drives, illustrate how the codeword is broken up and how the disk drives receive upper and lower 4-bit nibbles. Table 2 shows how, for codewords one through N, a different portion of the codeword is placed on the different drives. Each disk drive, for a given codeword, receives an upper and lower 4-bit nibble, designated with L's and U's, of the codeword. Additionally, the same section is used to store the nibbles on each of the disk drives used to store the codeword. In other words, for codeword_1, the first sector of disk drives $n-1$ through 0 receives the nibbles.

TABLE 2

	CODEWORD – DATA AND P AND Q				
	Sector of Drive d_{n-1}	Sector of Drive d_{n-2}	Sector of Drive d_0	Sector of Drive P	Sector of Drive Q
Codeword_1	Codeword_1 $(d_{n-1L})(d_{n-1U})$	Codeword_1 $(d_{n-2L})(d_{n-2U})$	Codeword_1 $(d_{0L})(d_{0U})$	Codeword_1 $(P_{1L})(P_{1U})$	Codeword_1 $(Q_{1L})(Q_{1U})$
Codeword_2	Codeword_2 $(d_{n-1L})(d_{n-1U})$	Codeword_2 $(d_{n-2L})(d_{n-2U})$	Codeword_2 $(d_{0L})(d_{0U})$	Codeword_2 $(P_{2L})(P_{2U})$	Codeword_2 $(Q_{2L})(Q_{2U})$

TABLE 2-continued

CODEWORD - DATA AND P AND Q					
	Sector of Drive d_{n-1}	Sector of Drive d_{n-2}	Sector of Drive d_0	Sector of Drive P	Sector of Drive Q
.					
.					
.					
Codeword_n	Codeword_n $(d_{n-1_L})(d_{n-1_U})$	Codeword_n $(d_{n-2_L})(d_{n-2_U})$	Codeword_n $(d_{0_L})(d_{0_U})$	Codeword_n $(P_{n_L})(P_{n_U})$	Codeword_n $(Q_{n_L})(Q_{n_U})$

Referring back to FIG. 10, for a parallel data write to the disks, the data is provided in parallel from buffers 330, 331. 332 and 333 along those data buses coupling the buffers to X-bar switches 310, 311, 312, and 313 after the 36-bits of data are disassembled in word assemblers 350 through 353 into 9-bit words. These X-bar switches are also coupled to inputs D3, D2, D1 an D0, respectively, of ACC 348 and ACC 349. In parallel processing modes, the two ACCs act as mutual "backups" to one another. Should all fail, the other will still perform the necessary error correcting functions. In addition to operating in a purely "backup" condition, the second level controller engine configures the ACCs so that each ACC is performing the error detection and correction functions for a portion of the set, the other ACC performing these functions for the remaining disk drives in the set. As the ACC units are still coupled to all the disk drives, failure of one or the other unit does not impact the system as the operating ACC can be reconfigured to act as the dedicated ACC unit for the entire set. For purposes of discussion, it is assumed here that ACC 348 is operating. ACC 348 will calculate the P and Q redundancy term for the data in the X-bar switches and provide the terms to its E_1 and E_2 outputs, which outputs are coupled to all the X-bar switches. For discussion only, it is assumed that only the E_2 connection of X-bar switch 314 and the E_1 connection of X-bar switch 315 are enabled. Thus, although the data is provided along the buses coupling ACC 348's E_1 and E_2 output to all the X-bar switches, the Q term is received only by X-bar switch 314 and the P term is received by X-bar switch 315. Then, the Q and P terms are provided first to DSI units 344 and 345 and then disk drives 20A5 and 20A6. It should be recalled that the various internal registers in the X-bar switches will act as a multi-stage pipeline, effectively slowing the transit of data through the switches sufficiently to allow ACC 348's redundancy circuit 302 to calculate the P and Q redundancy terms.

As ACC 349 is coupled to the X-bar switches in a substantially identical manner to ACC 348, the operation of the system when ACC 349 is operational is essentially identical to that described for ACC 348.

Subsequent parallel reads from the disks occur in the following manner. Data is provided on bi-directional buses to DSI units 340, 341, 342 and 343. P and Q redundancy terms are provided by DSI units 345 and 344, respectively. As the data and P and Q terms are being transferred through X-bar switches 310 through 315, ACC 348 uses the P and Q terms to determine if the data being received from the disk drives is correct. Word assemblers 350 through 353 assemble successive 9-bit words until the next 36-bits are available. This 36-bits are forwarded to buffers 330 through 333. Note that the 9-bit words are transmitted to the buffers in parallel. If that data is incorrect, the second level controller will be informed.

During a parallel read operation, in the event that there is a failure of a disk drive, the failed disk drive will, in certain instances, communicate to the second level controller that it has failed. The disk drive will communicate with the second level controller if the disk drive cannot correct the error using its own corrector. The second level controller will then communicate with ACCs 348 and 349 by loading the failed drive registers in the ACC (not shown in the figures) with the address of the failed drive. The failed drive can be removed from the set by deleting its address from the configuration registers. One of the set's spare drives can then be used in place of the failed drive by inserting the address of the spare drive into the configuration registers.

The ACC will then calculate the replacement data necessary to rewrite all the information that was on the failed disk onto the newly activated spare. In this invention, the term spare or backup drive indicates a disk drive which ordinarily does not receive or transmit data until another disk drive in the system has failed.

When the data, P, and Q bytes are received, the ACC circuits use the failed drive location in the failed drive registers to direct the calculation of the replacement data for the failed drive. After the calculation is complete. the data bytes, including the recovered data, are sent to data buffers in parallel. Up to two failed drives can be tolerated with the Reed-Solomon code implemented herein. All operations to replace failed disk drives and the data thereon occur when the system is operating in a parallel mode.

Regeneration of data occurs under second level controller control. When a failed disk drive is to be replaced, the ACC regenerates all the data for the replacement disk. Read/write operations are required until all the data has been replaced. The regeneration of the disk takes a substantial amount of time, as the process occurs in the background of the system's operations so as to reduce the impact to normal data transfer functions. Table 3 below shows the actions taken for regeneration reads. In Table 3, i represents a first failed drive and j represents a second failed drive. In Table 3, the column labelled Failed Drives indicates the particular drives that have failed. The last column describes the task of the ACC given the particular indicated failure.

TABLE 3

Failed Drives		Regeneration Read
P	—	ACC calculates P redundancy
Q	—	ACC calculates Q redundancy
i	—	ACC calculates replacement data for i drive
i	P	ACC calculates replacement data for i drive and P redundancy
Q	i	ACC calculates replacement data for i drive and Q redundancy
j	i	ACC calculates replacement data for i and j drives

TABLE 3-continued

Failed Drives		Regeneration Read
P	Q	ACC calculates P and Q redundancy

It should be noted that if both a data disk drive and a redundancy disk drive fail, the data on the data disk drive must be regenerated before the redundancy terms on the redundancy drive. During a regeneration write, regeneration data or redundancy terms are written to a disk and no action is required from the ACC logic.

During a parallel read operation, it should also be noted that additional error detection may be provided by the ACC circuitry.

Table 4 indicates what actions may be taken by the ACC logic unit when the indicated drive(s) has or have failed during a failed drive read operation. In this operation, the drives indicated in the Failed Drives columns are known to have failed prior to the read operation. The last column indicates the ACC response to the given failure.

TABLE 4

Failed Drives		
P		No action by ACC
Q		No action by ACC
i		ACC calculates replacement data
i	P	ACC calculates the replacement data
Q	i	ACC calculates the replacement data
i	j	ACC calculates replacement data
P	Q	No action by ACC

Transaction Processing Mode: Read

Transaction processing applications require the ability to access each disk drive independently. Although each disk drive is independent, the ACC codeword with P and Q redundancy is maintained across the set in the previously described manner. For a normal read operation, the ACC circuitry is not generally needed. If only a single drive is read, the ACC cannot do its calculations since it needs the data from the other drives to assemble the entire codeword to recalculate P and Q and compare it to the stored P and Q. Thus, the data is assumed to be valid and is read without using the ACC circuitry (see FIG. 15). Where drive 20C1 is the one selected, the data is simply passed through DSI unit 342 X-bar switch 312, word assembler 352 and buffer 332 to the external computer. If the disk drive has failed, the read operation is the same as a failed drive read in parallel mode with the exception that only the replacement data generated by the ACC is sent to the data buffer. In this case, the disk drive must notify the second level controller that it has failed, or the second level controller must otherwise detect the failure. Otherwise, the second level controller will not know that it should read all the drives, unless it assumes that there might be an error in the data read from the desired drive. The failed drive read is illustrated in FIG. 16, with drive 20C1 having the desired data, as in the example of FIG. 15. In FIG. 16, the second level controller knows that drive 20C1 has failed, so the second level controller calls for a read of all drives except the failed drive, with the drive 20C1 data being reconstructed from the data on the other drives and the P and Q terms. Only the reconstructed data is provided to its buffer, buffer 332, since this is the only data the external computer needs.

Transaction Processing Mode: Write

When any individual drive is written to, the P and Q redundancy terms must also be changed to reflect the new data (see FIG. 18). This is because the data being written over was part of a code word extending over multiple disk drives and having P and Q terms on two disk drives. The previously stored P and Q terms will no longer be valid when part of the codeword is changed, so new P and Q terms, P" and Q", must be calculated and written over the old P and Q terms on their respective disk drives. P" and Q" will then be proper redundancy terms for the modified code word.

One possible way to calculate P" and Q" is to read out the whole codeword and store it in the buffers. The new portion of the codeword for drive 20C1 can then be supplied to the ACC circuit along with the rest of the codeword, and the new P" and Q" can be calculated and stored on their disk drives as for a normal parallel write. However, if this method is used, it is not possible to simultaneously do another transaction mode access of a separate disk drive (i.e., drive 20A1) having part of the codeword, since that drive (20A1) and its buffer are needed for the transaction mode write for the first drive (20C1).

According to a method of the present invention, two simultaneous transaction mode accesses are made possible by using only the old data to be written over and the old P and Q to calculate the new P" and Q" for the new data. This is done by calculating an intermediate P' and Q' from the old data and old P and Q, and then using P' and Q' with the new data to calculate the new P" and Q". This requires a read-modify-write operation on the P and Q drives. The equations for the new P and Q redundancy are:

$$\text{New P redundancy } (P'') = (\text{old P} - \text{old data}) + \text{new data}$$

$$\text{New Q redundancy } (Q'') = (\text{old Q} - \text{old data} \cdot a_i) + \text{new data} \cdot a_i$$

$$P' = \text{old P} - \text{old data}$$

$$Q' = \text{old Q} - \text{old data} \cdot a_i$$

Where a_i is the coefficient from the syndrome equation S_1; and i is the index of the drive.

During the read portion of the read-modify-write, the data from the drive to be written to and the P and Q drives are summed by the ACC logic, as illustrated in FIG. 17. This summing operation produces the P' and Q' data. The prime data is sent to a data buffer. When the new data is in a data buffer, the write portion of the cycle begins as illustrated in FIG. 18. During this portion of the cycle, the new data and the P' and Q' data are summed by the ACC logic to generate the new P" and Q" redundancy. When the summing operation is complete, the new data is sent to the disk drive and the redundancy information is sent to the P and Q drives.

Parity Check of P and Q for Transaction Mode Write

During these read-modify-write operations, it is also possible that the ACC unit itself may fail. In this case, if the data in a single element were to be changed by a read-modify-write operation, a hardware failure in the

ACC might result in the redundancy bytes for the new data being calculated erroneously. To prevent this occurrence, the parity detector and parity generator are made part of the ACC circuitry. This additional redundant circuit is shown in FIGS. 14*a* and 14*b* and resides within redundancy circuit 302 as shown in FIG. 11. When data is received by the ACC circuitry, parity is checked to insure that no errors have occurred using the P and Q redundancy terms. In calculating Q'', new parity is generated for the product of the multiply operation and is summed with the parity of the old Q'' term. This creates the parity for the new Q term. For the P byte, the parity bits from the data are summed with the parity bit of the old P term to create the new parity bit for the new P'' term. Before writing the new data back to the disk drive, the parity of Q' (calculated as indicated previously) is checked. Should Q' be incorrect, the second level controller engine will be informed of an ACC failure. In this manner, a failure in the ACC can be detected.

The same operations are performed for a failed disk drive write in transaction processing operations as for parallel data writes, except that data is not written to a failed drive or drives.

With respect to transaction processing functions during normal read operations, no action is required from the ACC logic. The actions taken by the ACC logic during a failed drive read in transaction processing ode are listed in Table 5 below, where i and j represent the first and second failed drives. The columns labelled Failed Drives indicate which drives have failed. The last column indicates what action the ACC may or may not take in response to the indicated failure.

TABLE 5

Failed Drives		
P	—	Redundancy drives are not read; no ACC action
Q	—	Redundancy drives are not read; no ACC action
i	—	ACC logic calculates replacement data and performs a parallel read
i	P	ACC logic calculates replacement data and performs a parallel read
Q	i	ACC logic calculates replacement data and performs a parallel read
j	i	ACC logic calculates replacement data and performs a parallel read
P	Q	No ACC action as only data disk drives are read

If two data disk drives fail, the ACC logic must calculate the needed replacement data for both disk drives. If only one failed drive is to be read, both failed drives must still be noted by the ACC logic.

In the read-before-write operation (part of the read-modify-write process), the ACC logic generates P' and Q' redundancy terms. Table 6 shows the action taken by the ACC logic when a failed disk drive read precedes a write in this process. Again, i and j represent the first and second failed drives. The columns headed by Failed Drives indicate which drives have failed, and the last column denotes the response of the ACC to the indicated failures.

TABLE 6

Failed Drives		
P	—	ACC calculates Q' only
Q	—	ACC calculates P' only
i	—	ACC logic takes no action and all good data disk drives are read into data buffers
i	P	All good data disk drives are read into data buffers
Q	i	All good data disk drives are read into data buffers
i	j	All good data disk drives are read into data buffers
i	failed drive	Perform a parallel read, the ACC logic calculates the replacement data for the jth failed drive. Next, the remaining good data disk drives are read into the data buffers.
P	Q	No read before write operation is necessary

When a failed data disk drive is to be written, all good data disk drives must be read so that a new P and Q redundancy can be generated. All of the data from the good data disk drive and the write data is summed to generate the new redundancy. When two data disk drives fail, the ACC logic must calculate replacement data for both failed drives. If only one drive is to be read, both must be reported to the ACC logic.

During write operations, the ACC continues to calculate P and Q redundancy. Table 7 shows the ACC's tasks during failed drive writes. Here P and Q represent the P and Q redundancy term disk drives, and i and j represent the first and second failed data disk drives. The columns Failed Drives denote the particular failed drives, and the last column indicates the ACC response to the failed drives.

TABLE 7

Failed Drives		
P	—	ACC calculates Q redundancy only
Q	—	ACC calculates P redundancy only
i	—	ACC calculates P and Q redundancy
i	P	ACC calculates Q redundancy only
Q	i	ACC calculates P redundancy only
i	j	ACC calculates P and Q redundancy
P	Q	ACC logic takes no action

Summary of ECC

The interconnected arrangements herein described relative to both preferred embodiments of the present invention allow for the simultaneous transmission of data from all disks to the word assemblers or vice versa. Data from or to any given disk drive may be routed to any other word assembler through the X-bar switches under second level controller engine control. Additionally, data in any word assembler may be routed to any disk drive through the X-bar switches. The ACC units receive all data from all X-bar switches simultaneously. Any given disk drive, if it fails, can be removed from the network at any time. The X-bar switches provide alternative pathways to route data or P and Q terms around the failed component.

The parallel arrangement of disk drives and X-bar switches creates an extremely fault-tolerant system. In the prior art, a single bus feeds the data from several disk drives into a single large buffer. In the present invention, the buffers are small and one buffer is assigned to each disk drive. The X-bar switches, under control of the ACC units, can route data from any given disk drive to any given buffer and vice versa. Each second level controller has several spare disks and one spare buffer coupled to it. The failure of any two disks can be easily accommodated by switching the failed disk from the configuration by means of its X-bar switch and switching one of the spare disks onto the network. The present invention thus uses the error detection and correction capabilities of a Reed-Solomon error correction code in an operational environment where the system's full operational capabilities can be maintained by reconfiguring the system to cope with any detected disk or buffer failure. The ACC can correct and regenerate the data for the failed disk drive and, by reconfiguring the registers of the failed and spare disk drives, effectively remove the failed drive from the system and regenerate or reconstruct the data from the failed disk onto the spare disk.

Disk Drive Configuration and Format

The present invention allows a set of physical mass data storage devices to be dynamically configured as one or more logical mass storage devices. In accordance with the present invention, such a set of physical devices is configurable as one or more redundancy groups and each redundancy group is configurable as one or more data groups.

A redundancy group, as previously used in known device sets, is a group of physical devices all of which share the same redundant device set. A redundant device is a device that stores duplicated data or check data for purposes of recovering stored data if one or more of the physical devices of the group fails.

Where check data is involved, the designation of a particular physical device as a redundant device for an entire redundancy group requires that the redundant device be accessed for all write operations involving any of the other physical devices in the group. Therefore, all write operations for the group interfere with one another, even for small data accesses that involve less than all of the data storage devices.

It is known to avoid this contention problem on write operations by distributing check data throughout the redundancy group, thus forming a logical redundant device comprising portions of several or all devices of the redundancy group. For example, FIG. 19 shows a group of 13 disk storage devices. The columns represent the various disks D1-D13 and the rows represent different sectors S1-S5 on the disks. Sectors containing check data are shown as hatched. Sector S1 of disk D13 contains check data for sectors of disks D1-D12. Likewise, the remaining hatched sectors contain check data for their respective sector rows. Thus, if data is written to sector S4 of disk D7, then updated check data is written into sector S4 of disk D10. This is accomplished by reading the old check data, re-coding it using the new data, and writing the new check data to the disk. This operation is referred to as a read-modify-write. Similarly, if data is written to sector S1 of disk D11, then check data is written into sector S1 of disk D13. Since there is no overlap in this selection of four disks for writes, both read-modify-write operations can be performed in parallel.

A distribution of check data in a redundancy group in the manner shown in FIG. 19 is known as a striped check data configuration. The term "striped redundancy group" will be used herein to refer generally to a redundancy group in which check data is arranged in a striped configuration as shown in FIG. 19, and the term "redundancy group stripe depth" will be used herein to refer to the depth of each check data stripe in such a striped redundancy group.

In previously known device sets, it was known to provide the whole set as a single redundancy group. It has been found that a redundancy group can be divided

into various "extents", each defined as a portion of the depth of the redundancy group and each capable of having a configuration of check data different from that of other extents in the same redundancy group. Moreover, it has been found that more than ne redundancy group can be provided in a single device set, under the control of a single "array controller", and connected to a main processing unit via one or more device controllers.

Similarly, in previously known device sets, the single redundancy group included only one data group for application data—i.e., the device set operated as a single logical device. It has been found, however, that a redundancy group can be broken up into multiple data groups, each of which can operate as a separate logical storage device or as part of a larger logical storage device. A data group can include all available mass storage memory on a single physical device (i.e., all memory on the device available for storing application data), or it can include all available mass storage memory on a plurality of physical devices in the redundancy group. Alternatively, as explained more fully below, a data group can include several physical devices, but instead of including all available mass storage memory of each device might only include a portion of the available mass storage memory of each device. In addition, it has been found that it is possible to allow data groups from different redundancy groups to form a single logical device. This is accomplished, as will be more fully described, by superimposing an additional logical layer on the redundancy and data groups.

Moreover, in previously known device sets in which application data is interleaved across the devices of the set, the data organization or geometry is of a very simple form. Such sets generally do not permit different logical organizations of application data in the same logical unit nor do they permit dynamic mapping of the logical organization of application data in a logical unit. It has been found that the organization of data within a data group can be dynamically configured in a variety of ways. Of particular importance, it has been found that the data stripe depth of a data group can be made independent of redundancy group stripe depth, and can be varied from one data group to another within a logical unit to provide optimal performance characteristics for applications having different data storage needs.

An embodiment of a mass storage system **500** including two second level controllers **14A** and **14B** is shown in the block diagram of FIG. 20. As seen in FIG. 20, each of parallel sets **501** and **502** includes thirteen physical drives **503-515** and a second level controller **14**. Second level controller **14** includes a microprocessor **516a** which controls how data is written and validated across the drives of the parallel set. Microprocessor **516a** also controls the update or regeneration of data when one of the physical drives malfunctions or loses synchronization with the other physical drives of the parallel set. In accordance with the present invention, microprocessor **516a** in each second level controller **14** also controls the division of parallel sets **501** and **502** into redundancy groups, data groups and application units. The redundancy groups, data groups and application units can be configured initially by the system operator when the parallel set is installed, or they can be configured at any time before use during run-time of the parallel set. Configuration can be accomplished, as described in greater detail below, by defining certain configuration parameters that are used in creating various address maps in the program memory of microprocessor 516a and, preferably, on each physical drive of the parallel set.

Each of second level controllers 14A and 14B is connected to a pair of first level controllers 12A and 12B. Each first level c ntr ller is in turn connected by a bus or channel 522 to a CPU main memory. In general, each parallel set is attached to at least two sets of controllers so that there are at least two parallel paths from one or more CPU main memories to that parallel set. Thus, for example, each of the second level controllers 14A and 14B is connected to first level controllers 12A and 12B by buses 524 and 526. Such parallel data paths from a CPU to the parallel set are useful for routing data around a busy or failed first or second level controllers as described above.

Within each parallel set are an active set 528 comprising disk drive units 503-514, and a backup set 530 comprising disk drive unit 515. Second level controller 14 routes data between first level controllers 12 and the appropriate one or ones of disk drive units 503-515. First level controllers 12 interface parallel sets 501 and 502 to the main memories of one or more CPUS; and are responsible for processing I/O requests from applications being run by those CPUs. A further description of various components of the apparatus of parallel sets 501 and 502 and first level controllers 12 can be found in the following co-pending, commonly assigned U.S. patent applications incorporated herein in their entirety by reference: Ser. No. 07/487,648 entitled "NON-VOLATILE MEMORY STORAGE OF WRITE OPERATION IDENTIFIER IN DATA STORAGE DEVICE," filed in the names of David T. Powers, Randy Katz, David H. Jaffe, Joseph S. Glider and Thomas E. Idleman; and Ser. No. 07/488,750 entitled "DATA CORRECTIONS APPLICABLE TO REDUNDANT ARRAYS OF INDEPENDENT DISKS," filed in the names of David T. Powers, Joseph S. Glider and Thomas E. Idleman.

To understand how data is spread among the various physical drives of an active set 528 of a parallel set 501 or 502, it is necessary to understand the geometry of a single drive. FIG. 21 shows one side of the simplest type of disk drive—a single platter drive. Some disk drives have a single disk-shaped "platter" on both sides of which data can be stored. In more complex drives, there may be several platters on one "spindle," which is the central post about which the platters spin.

As shown in FIG. 21, each side 600 of a disk platter is divided into geometric angles 601, of which eight are shown in FIG. 21, but of which there could be some other number. Side 600 is also divided into ring-shaped "tracks" of substantially equal width, of which seven are shown in FIG. 21. The intersection of a track and a geometric angle is known as a sector and typically is the most basic unit of storage in a disk drive system. There are fifty-six sectors 603 shown in FIG. 21.

A collection of tracks 602 of equal radius on several sides 600 of disk platters on a single spindle make up a "cylinder." Thus, in a single-platter two-sided drive, there are cylinders of height=2, the number of cylinders equalling the number of tracks 602 on a side 600. In a two-platter drive, then, the cylinder height would be 4. In a one-sided single-platter drive, the cylinder height is 1.

A disk drive is read and written by "read/write heads" that move over the surfaces of sides 600. FIG. 22 shows the distribution of data sub-units—sectors, tracks and cylinders—in a group 716 of eight single-platter two-sided drives 700-707 in a manner well-suited to illustrate the present invention. Drives 700-707 may, for example, correspond to drive units 503-510 of parallel set 501 or 502. Each of the small horizontal divisions represents a sector 708. For each drive, four cylinders 709-712 are shown, each cylinder including two tracks 713 and 714, each track including five sectors.

In the preferred embodiment shown in FIG. 22, group 716 comprises a single redundancy group in which two types of redundancy data, referred to as "P" check data and "Q" check data, are used to provide data redundancy. The P and Q check data are the results of a Reed-Solomon coding algorithm applied to the mass storage data stored within the redundancy group. The particular method of redundancy used is implementation specific. As shown, the redundancy data is distributed across all spindles, or physical drives, of group 716, thus forming two logical check drives for the redundancy group comprising group 716. For example, the P and Q check data for the data in sectors 708 of cylinders 709 of drives 700-705 are contained respectively in cylinders 709 of drives 706 and 707. Each time data is written to any sector 708 in any one of cylinders 709 of drives 700-705, a read-modify-write operation is performed on the P and Q check data contained in corresponding sectors of drives 706 and 707 to update the redundancy data.

Likewise, cylinders 710 of drives 700-707 share P and Q check data contained in cylinders 710 of drives 704 and 705; cylinders 711 of drives 700-707 share P and Q check data contained in cylinders 711 of drives 702 and 703; and cylinders 712 of drives 700-707 share P and Q check data contained in cylinders 712 of drives 700 and 701.

Three data groups D1-D3 are shown in FIG. 22. Data group D1 includes cylinders 709 of each of spindles 700, 701. Data group D2 includes cylinders 709 of each of spindles 702, 703. Data group D3 includes all remaining cylinders of spindles 700-707, with the exception of those cylinders containing P and Q check data. Data group D1 has a two-spindle bandwidth, data group D2 has a four-spindle bandwidth and data group D3 has a six-spindle bandwidth. Thus it is shown in FIG. 22 that, in accordance with the principles of the present invention, a redundancy group can comprise several data groups of different bandwidths. In addition, each of data groups D1-D3 may alone, or in combination with any other data group, or groups, comprise a separate logical storage device. This can be accomplished by defining each data group or combination as an individual application unit. Application units are discussed in greater detail below.

In FIG. 22, sectors 708 are numbered within each data group as a sequence of logical data blocks. This sequence is defined when the data groups are configured, and can be arranged in a variety of ways. FIG. 22 presents a relatively simple arrangement in which the sectors within each of data groups D1-D3 are numbered from left to right in stripes crossing the width of the respective data group, each data stripe having a depth of one sector. This arrangement permits for the given bandwidth of each data group a maximum parallel transfer rate of consecutively numbered sectors.

The term "data group stripe depth" is used herein to describe, for a given data group, the number of logically contigu us data sectors stored on a drive within the boundaries of a single stripe of data in that data group.

In accordance with the principles of the present invention, the depth f a data group stripe may be lesser than, greater than or equal to the depth f redundancy group stripe. As one example of this, FIG. 22 shows that data groups D1-D3 each has a data group stripe depth of ne sector, and are all included in a redundancy group having a redundancy group stripe depth of one cylinder.

Redundancy group 716 can handle up to six data read requests simultaneously—one from each of spindles 700-705—because the read/write heads of the spindles can move independently of one another. Redundancy group 716 as configured in FIG. 22 also can handle certain combinations of write requests simultaneously. For example, in many instances any data sector of data group D1 can be written simultaneously with any data sectors of data group D3 contained on spindles 702-705 that are not backed up by P or Q check data on spindles 700, 701, 706 or 707.

Redundancy group 716 as configure#in FIG. 22 usually cannot handle simultaneous write operations to sectors in data groups D1 and D2, however, because to perform a write operation in either of these data groups, it is necessary to write to drives 706 and 707 as well. Only one write operation can be performed on the check data of drives 706, 707 at any one time, because the read/write heads can only be in one place at one time. Likewise, regardless of the distribution of data groups, write operations to any two data sectors backed up by check data on the same drive cannot be done simultaneously. The need for the read/write heads of the check drives to be in more than one place at one time can be referred to as "collision."

It is to be understood that the above-described restriction concerning simultaneous writes to different data drives sharing common check drives is peculiar to check drive systems, and is not a limitation of the invention. For example, the restriction can be avoided by implementing the invention using a mirrored redundancy group, which does not have the property that different data drives share redundancy data on the same drive.

FIG. 23 shows a more particularly preferred embodiment of redundancy group 716 configured according to the present invention. In FIG. 23, as in FIG. 22, the logical check "drives" are spread among all of spindles 700-707 on a per-cylinder basis, although they could also be on a per-track basis or even a per-sector basis.

Data groups D1 and D2 are configured as in FIG. 22. The sectors of data group D3 of FIG. 22, however, have been divided among four data groups D4-D7. As can be seen in FIG. 23, the sequencing of sectors in data groups D4-D7 is no longer the same as the single-sector-deep striping of data groups D1 and D2. Data group D4 has a data group stripe depth of 20 sectors—equal to the depth of the data group itself. Thus, in data group D4 logically numbered sectors 0-19 can be read consecutively by accessing only a single spindle 700, thereby allowing the read/write heads of spindles 701-707 to handle other transactions. Data groups D5, D6 and D7 each show examples of different intermediate data group stripe depths of 5 sectors, 2 sectors and 4 sectors, respectively.

The distribution of the check data over the various spindles can be chosen in such a way as to minimize collisions. Further, given a particular distribution, then to the extent that second level controller 14 has a choice in the order of operations, the order can be chosen to minimize collisions.

The distribution f redundancy groups and data groups over the active set 528 of a parallel set 501 or 502 can be parameterized. For example, the redundancy group can be characterized by a redundancy group width (in spindles), representing the number of spindles spanned by a particular set of check data, a redundancy group depth (in any subunit—sector, track or cylinder) and a redundancy group stripe depth (also in any subunit—sector, track or cylinder). Data groups can be characterized by width (in spindles), depth (in any subunit—sector, track or cylinder), and data group stripe depth (also in any subunit sector, track or cylinder). Because data groups do not start only at the beginning of active set 528, they are also characterized by a "base", which is a two-parameter indication of the spindle and the offset from the beginning of the spindle at which the data group starts. A redundancy group may, like a data group, include less than all of an entire spindle. In addition, as previously stated herein, a redundancy group may be divided into a plurality of extents. The extents of a redundancy group have equal widths and different bases and depths. For each extent, the distribution of check data therein can be independently parameterized. In the preferred embodiment, each redundancy group extent has additional internal parameters, such as the depth of each redundancy group stripe within the redundancy group extent and the drive position of the P and Q check data for each such redundancy group stripe.

Redundancy group width reflects the trade-off between reliability and capacity. If the redundancy group width is high, then greater capacity is available, because only two drives out of a large number are used for check data, leaving the remaining drives for data. At the other extreme, if the redundancy group width=4, then a situation close to mirroring or shadowing, in which 50% of the drives are used for check data, exists (although with mirroring if the correct two drives out of four fail, all data on them could be lost, while with check data any two drives could be regenerated in that situation). Thus low redundancy group widths represent greater reliability, but lower capacity per unit cost, while high redundancy group widths represent greater capacity per unit cost with lower, but still relatively high, reliability.

Data group width reflects the trade-off discussed above between bandwidth and request rate, with high data group width reflecting high bandwidth and low data group width reflecting high request rates.

Data group stripe depth also reflects a trade-off between bandwidth and request rate. This trade-off varies depending on the relationship of the average size of I/O requests to the data group and the depth of data stripes in the data group. The relationship of average I/O request size to the data group stripe depth governs how often an I/O request to the data group will span more than one read/write head within the data group; it thus also governs bandwidth and request rate. If high bandwidth is favored, the data group stripe depth is preferably chosen such that the ratio of average I/O request size to stripe depth is large. A large ratio results in I/O requests being more likely to span a plurality of data drives, such that the requested data can be accessed at a higher bandwidth than if the data were located all on one drive. If, on the other hand, a high request rate is favored, the data group stripe depth is preferably chosen such that the ratio of I/O request size to data group stripe depth is small. A small ratio results in a lesser likelihood that an I/O request will span more than one data drive, thus increasing the likelihood that multiple I/O requests to the data group can be handled simultaneously.

The variance of the average size of I/O requests might also be taken int account in choosing data group stripe depth.. For example, for a given average I/O request size, the data group stripe depth needed to achieve a desired request rate might increase with an increase in I/O request size variance.

In accordance with the present invention, the flexibility of a mass storage apparatus comprising a plurality of physical mass storage devices can be further enhanced by grouping data groups from one or from different redundancy groups into a common logical unit, referred to herein as an application unit. Such application units can thus appear to the application software of an operating system as a single logical mass storage unit combining the different operating characteristics of various data groups. Moreover, the use of such application units permits data groups and redundant groups to be configured as desired by a system operator independent of any particular storage architecture expected by application software. This additional level of logical grouping, like the redundancy group and data group logical levels, is controlled by second level controller 14.

FIG. 24 illustrates an example of how application units, data groups and redundancy groups might be mapped to a device set such as parallel set 501 or 502, at initialization of the parallel set.

Referring first to the linear graph 800 of logical unit address space, this graph represents the mass data storage memory of the parallel set as it would appear to the application software of a CPU operating system. In the particular example of FIG. 24, the parallel set has been configured to provide a logical unit address space comprising two application (logical) units (LUN0 and LUN1). Logical unit LUN0 is configured to include 20 addressable logical blocks having logical block numbers LBN0-LBN19. As shown by FIG. 24, logical unit LUN0 also includes an unmapped logical address space 802 that is reserved for dynamic configuration. Dynamic configuration means that during run-time of the parallel set the CPU application software can request to change the configuration of the parallel set from its initial configuration. In the example of FIG. 24, unmapped spaces 802 and 804 are reserved respectively in each of logical units LUN0 and LUN1 to allow a data group to be added to each logical unit without requiring that either logical unit be taken off line. Such dynamic configuration capability can be implemented by providing a messaging service for a CPU application to request the change in configuration. On behalf of mass storage system 500, the messaging service can be handled, for example, by the first level controllers 12. Logical unit LUN1 includes a plurality of addressable logical blocks LBN0-LBN179 and LBN200-LBN239. The logical blocks LBN180-LBN199 are reserved for dynamic configuration, and in the initial configuration of the parallel set, as shown in FIG. 24, are not available to the application software.

The mass storage address space of logical unit LUN0 comprises a single data group D1, as shown by data group address space chart 806. Data group D1 includes 20 logically contiguous data blocks 0-19, configured as shown in FIG. 22 and corresponding one to one with logical block numbers LBN0-LBN19. Logical unit LUN1 includes two data groups D2 and D3, comprising respectively 40 data blocks numbered 0-39 corresponding to logical blocks LBN200-239 of logical unit LUN1, and 180 data blocks numbered 0-179 corresponding to logical blocks LBN0-LBN179 of logical unit LUN1. As shown by the example of FIG. 24, the logical blocks of a logical unit can be mapped as desired to the data blocks of one or more data groups in a variety of ways. Data group address space 806 also includes additional data groups (D4) and (D5) reserved for dynamic configuration. These data groups can be formatted on the disk drives of the parallel set at initialization or at any time during the run-time of the parallel set, but are not available to the application software in the initial configuration of the parallel set.

The redundancy group configuration of the parallel set is illustrated by a two dimensional address space 808, comprising the entire memory space of the parallel set. The horizontal axis of address space 808 represents the thirteen physical drives of the parallel set, including the twelve drives of active set 528 and the one spare drive of backup set 530. In FIG. 24, the drives of the active set are numbered 0-11 respectively to reflect their logical positions in the parallel set. The vertical axis of address space 808 represents the sectors of each physical drive. As shown by redundancy group address space 808, the parallel set has been configured as one redundancy group RG0 having three extents A, B and C. As can be seen, the width of each extent is equal to that of the redundancy group RG0: 12 logical drive positions or, from another perspective, the entire width of active set 528.

Extent A of redundancy group RG0 includes sectors 1-5 of drives 0-11. Thus, extent A of redundancy group RG0 has a width of 12 spindles, and an extent depth of 5 sectors. In the example of FIG. 24, extent A is provided as memory space for diagnostic programs associated with mass storage system 500. Such diagnostic programs may configure the memory space of extent A in numerous ways, depending on the particular diagnostic operation being performed. A diagnostic program may, for example, cause a portion of another extent to be reconstructed within the boundaries of extent A, including application data and check data.

Extent B of redundancy group RG0 includes all application data stored on the parallel set. More particularly, in the example of FIG. 24, extent B includes data groups D1, D2 and D3 configured as shown in FIG. 22, as well as additional memory space reserved for data groups (D4) and (D5), and a region 809 of memory space not mapped to either logical unit LUN0 or LUN1. This region 809 may, for example, be mapped to another logical unit (e.g., LUN2) being used by another application.

Address space 808 also includes a third extent C in which a second diagnostic field may be located. Although the parallel set is shown as including only a single redundancy group RG0, the parallel set may alternatively be divided into more than one redundancy group. For example, redundancy group RG0 might be limited to a width of 8 spindles including logical drive positions 0-7, such as is shown in FIGS. 22 and 23, and a second redundancy group might be provided for logical drive positions 8-11.

It is also not necessary that the entire depth of the parallel set be included in redundancy group RG0. As an example, FIG. 24 shows that above and below redundancy group RG0 are portions 810 and 811 of memory space 808 that are not included in the redundancy group. In the example of FIG. 24, portions 810 and 811 contain data structures reflecting the configuration of the parallel set. These data structures are described in greater detail below in connection with FIG. 25. In addition, any portion f memory space between set extents A, B and C, such as the portions indicated by regions D and E in FIG. 24, may be excluded from redundancy group RG0.

FIG. 24 further provides a graph 812 showing a linear representation of the physical address space of the drive in logical position 0. Graph 812 represents a sectional view of address space chart 810 along line 0'-0", and further illustrates the relationship of the various logical levels of the present invention as embodied in the exemplary parallel set configuration of FIG. 24.

As stated previously, the parallel set can be configured by the operator initially at installation time and/or during run-time of the parallel set. The operator formats and configures the application units he desires to use by first determining the capacity, performance and redundancy requirements for each unit. These considerations have been previously discussed herein. Once the capacity, performance and redundancy requirements have been defined, the logical structure of the units can be specified by defining parameters for each of the logical layers (redundancy group layer, data group layer and application unit layer). These parameters are provided to a configuration utility program executed by processor 516*a* of second level controller 14. The configuration utility manages a memory resident database of configuration information for the parallel set. Preferably, a copy of this database information is kept in nonvolatile memory to prevent the information from being lost in the event of a power failure affecting the parallel set. A format utility program executed by processor 516*a* utilizes the information in this database as input parameters when formatting the physical drives of the parallel set as directed by the operator.

The basic parameters defined by the configuration database preferably include the following:

1) For each redundancy group:	

Type:	Mirrored;
	Two check drives:
	One check drive;
	No check drive.

Width:	The number of logical drive positions as spindles in the redundancy group.
Extent Size:	For each extent of the redundancy group, the size (depth) of the extent in sectors
Extent Base:	For each extent of the redundancy group the physical layer address of the first sector in the extent.
Stripe in Depth:	For interleaved check drive groups, the depth, sectors of a stripe of check data.
Drives:	An identification of the physical drives included in the redundancy group.
Name:	Each redundancy group has a name that is unique across the mass storage system 500.
2) For each data group:	
Base:	The index (logical drive number) of the drive position within the redundancy group that is the first drive position in the data group within the redundancy group.
Width:	The number of drive positions (logical drives) in the data group. This is the number of sectors across in the data group address space.
Start:	The offset, in sectors, within the redundancy group extent where the data group rectangle begins on the logical drive position identified by the base parameter.
Depth:	The number of sectors in a vertical column of the data group, within the redundancy group extent. Depth and width together are the dimensions respectively of the side and the top of the rectangle formed by each data group as shown in FIGS. 22-24.
Redundancy Group:	The name of the redundancy group to which the data group belongs.
Extent Number:	A name or number identifying the extent in which the data group is located.
Index:	The configuration utility will assign a number to each data group, unique within its redundancy group. This number will be used to identify the data group later, for the format utility and at run-time.
Data Group Stripe Depth:	The depth, in sectors, of logically contiguous blocks of data within each stripe of data in the data group.
3) For each application unit:	
Size:	Size in sectors
Data Group List:	A list of the data groups, and their size and order, within the unit address space, and the base unit logical address of each data group. Each group is identified by the name of the redundancy group it is in and its index.

FIG. 25 illustrates exemplary data structures containing the above-described parameters that can be used in implementing the configuration database of a device set such as parallel set 501 or 502. These data structures may be varied as desired to suit the particular device set embodiment to which they are applied. For example, the data structures described hereafter allow for many options that may be unused in a particular device set, in which case the data structures may be simplified.

The configuration database includes an individual unit control block (UCB) for each application unit that references the parallel set (a unit may map into more than one parallel set). These UCB's are joined together in a linked list 900. Each UCB includes-a field labeled APPLICATION UNIT # identifying the number of the application unit described by that UCB. Alternatively, the UCB's within link list 900 might be identified by a table of address pointers contained in link list 900 or in some other data structure in the program memory of microprocessor 516*a*. Each UCB further includes a map 901 of the data groups that are included in that particular application unit. Data group map 901 includes a count field 902 defining the number of data groups within the application unit, a size field 904 defining the size of the application unit in sectors, and a type field 906 that defines whether the linear address space of the application unit is continuous (relative addressing) or non-continuous (absolute addressing). A non-continuous address space is used to allow portions of the application unit to be reserved for dynamic configuration as previously described in connection with data groups (D4) and (D5) of FIG. 22.

Data group map 901 further includes a data group mapping element 908 for each data group within the application unit. Each data group mapping element 908 includes a size field 910 defining the size in sectors of the corresponding data group, a pointer 912 to a descriptor block 914 within a data group list 916, a pointer 718 to an array control block 720, and an index field 721. The data group mapping elements 908 are listed in the order in which the data blocks of each data group map to the LBN's of the application unit. For example, referring to LUN1 of FIG. 24, the mapping element for data group D3 would be listed before the data group mapping element for data group D2. Where the address space of the application unit is non-continuous, as in the case of LUN1 of FIG. 24, data group map 901 may include mapping elements corresponding to, and identifying the size of, the gaps between available ranges of LBN's.

Data group list 916 includes a descriptor block 914 for each data group within the parallel set, and provides parameters for mapping each data group to the redundancy group and redundancy group extent in which it is located. Data group list 916 includes a count field 717 identifying the number of descriptor blocks in the list. In the case of a redundancy group having a striped check data configuration, each data group descriptor block 914 may include a "pqdel" field 722 that defines the offset of the first data block of the data group from the beginning of the check data for the redundancy group stripe that includes that first data block. The value of pqdel field 722 may be positive or negative, depending on the relative positions of the drive on which the first data block of the data group is configured and the corresponding check data drives for the redundancy group stripe including that first data block. This value can be useful for assisting the second level controller in determining the position of the check data during I/O operations.

Each data group descriptor block 914 also includes an index field 723 (same value as index field 721), a width field 724, a base field 726, an extent number field 727, a start field 728, a depth field 730, a data group stripe depth field 731 and a redundancy group name field 732 that respectively define values for the corresponding parameters previously discussed herein.

Array control block 720 provides a map of redundancy groups of the parallel set to the physical address space of the drives comprising the parallel set. Array control block 720 includes an array name field 734 and one or more fields 735 that uniquely identify the present configuration of the parallel set. Array control block 720 also includes a list of redundancy group descriptor blocks 736. Each redundancy group descriptor block 736 includes a redundancy group name field 738 identifying the redundancy group corresponding to the descriptor block, a redundancy group width field 740 and a redundancy group extent map 742. Array control block 720 further includes a list 744 of physical drive identifier blocks 745.

For each extent within the redundancy group, redundancy group extent map 742 includes an extent descriptor block 746 containing parameters that map the extent to corresponding physical address in the memory space of the parallel set, and define the-configuration of redundant information in the extent. As an example, extent descriptor blocks are shown for the three extents of redundancy group RG0 of FIG. 24, each extent descriptor block including an extent number field 747 and base and size fields defining the physical addresses of the corresponding extent. Application data base and size fields 748 and 750 correspond respectively to the base and size of extent B of redundancy group RG0; diagnostic (low) base and size fields 752 and 754 correspond respectively to the base and size of extent A of redundancy group RG0; and diagnostic (high) base and size fields 756 and 758 correspond respectively to the base and size of extent C of redundancy group RG0.

Each extent descriptor block 746 also includes a type field 760 that defines the type of redundancy implemented in the extent. For example, a redundancy group extent may be implemented by mirroring or shadowing the mass storage data stored in the data group(s) within the extent (in which case, the extent will have an equal number of data drives and redundant drives). Alternatively, a Reed-Solomon coding algorithm may be used to generate check data on one drive for each redundancy group stripe within the extent, or a more sophisticated Reed-Solomon coding algorithm may be used to generate two drives of check data for each redundancy group stripe. Type field 760 may specify also whether the check data is to be striped throughout the extent, and how it is to be staggered (e.g., the type field might index a series of standardized check data patterns, such as a pattern in which check data for the first redundancy group stripe in the extent is located on the two numerically highest logical drive positions of the redundancy group, check data for the second redundancy group stripe in the extent is located on the next two numerically highest logical drive positions, and so on). Yet another alternative is that type field 760 indicates that no check drives are included in the initial configuration of the redundancy group extent. This may be desired, for example, if the redundancy group extent is created for use by diagnostic programs. A redundancy group extent of this type was previously discussed in connection with extent A of redundancy group RG0 shown in FIG. 24.

Each extent descriptor block 746 may further include a redundancy group stripe depth field 762 to specify, if appropriate, the depth of redundancy group stripes within the extent.

List 744 of physical drive identifier blocks 745 includes an identifier block 745 for each physical drive in the parallel set. Each identifier block 745 provides information concerning the physical drive and its present operating state, and includes in particular one or more fields 764 for defining the logical position in the parallel set of the corresponding physical drive.

To summarize briefly the intended functions of the various data structures of FIG. 25, the unit control blocks of link list 900 define the mapping of application units to data groups within the parallel set. Mapping of data groups to redundancy groups is defined by data group list 916, and mapping of redundancy groups to the physical address space of the memory of the parallel set is defined by array control block 720.

When each physical disk of the parallel set is formatted by the formatting utility, a copy of the array control block 720, link list 900 and data group list 916 are stored on the drive. This information may be useful for various operations such as reconstruction of a failed drive. A copy of the configuration database also may be written to the controller of another parallel set, such that if one parallel set should fail, another would be prepared to take its place.

During each I/O request to a parallel set, the mapping from unit address to physical address spaces must be made. Mapping is a matter of examining the configuration database to translate: (1) from a unit logical address span specified in the I/O request to a sequence of data group address spans; (2) from the sequence of data group address spans to a set of address spans on logical drive positions within a redundancy group; and then (3) from the set of address spans on logical drive positions to actual physical drive address spans. This mapping process can be done by having an I/O request server step through the data structures of the configuration database in response to each I/O request. Alternatively, during initialization of the parallel set the configuration utility may, in addition to generating the configuration database as previously described, generate subr utines for the I/O request server for performing a fast mapping function unique to each data group. The particular manner in which the I/O request server carries out the mapping operations is implementation specific, and it is believed to be within the skill of one in the art t implement an I/O request server in accordance with the present invention as the invention is described herein.

The following is an example of how the I/O request server might use the data structures of FIG. 25 to map from a logical unit address span of an application I/O request to a span or spans within the physical address space of a parallel set. The logical unit address span is assumed to be defined in the I/O request by a logical application unit number and one or more LBN's within that application unit.

The I/O request server determines from the I/O request the application unit being addressed and whether that application unit references the parallel set. This latter determination can be made by examining link list 900 for a UCB having an APPLICATION UNIT # corresponding to that of the I/O request. If an appropriate UCB is located, the I/O request server next determines from the LBN(S) specified in the I/O request the data group or data groups in which data block(s) corresponding to those LBN(S) are located. This can be accomplished by comparing the LBN(S) to the size fields 910 of the mapping elements in data group map 901, taking into account the offset of that size field from the beginning of the application unit address space (including any gaps in the application unit address space). For example, if the size value of the first data group mapping element in map 901 is greater than the LBN(s) of the I/O request, then it is known that the LBN(s) correspond to data blocks in that data group. If not, then the size value of that first mapping element is added to the size value of the next mapping element in map 901 and the LBN(s) are checked against the resulting sum. This process is repeated until a data group is identified for each LBN in the I/O request.

Having identified the appropriate data group(s), the I/O request server translates the span of LBN's in the I/O request into one or more spans of corresponding data block numbers within the identified data group(s). The configuration utility can then use the value of index field 921 and pointer 912 within each mapping element 908 corresponding to an identified data group to locate the data group descriptor block 914 in data group list 916 for that data group. The I/O request server uses the parameters of the data group descriptor block to translate each span of data block numbers into a span of logical drive addresses.

First, the I/O request server determines the logical drive position of the beginning of the data group from the base field 726 of the data group descriptor block 914. The I/O request server also determines from fields 732 and 727 the redundancy group name and extent number in which the data group is located, and further determines from start field 728 the number of sectors on the drive identified in base field 726 between the beginning of that redundancy group extent and the beginning of the data group. Thus, for example, if the I/O request server is reading the descriptor block for data group D3 configured as shown in FIG. 24, base field 726 will indicate that the data group begins on logical drive position 0, redundancy name field 732 will indicate that the data group is in redundancy group RG0, extent field 727 will indicate that the data group is in extent B, and

start field 728 will indicate that there is an offset of 10 sectors on l gical drive 0 between the beginning of extent B and the first data block of data group D3.

Knowing the logical drive position and extent ffset of the first data block of the data group, the I/O request server then determines the l gical drive positi n and extent offset for each sequence of data blocks in the data group corresponding to the LBN's of the I/O request. To do this, the I/O request server may use the values of width field 724, depth field 730 and data group stripe depth field 731. If any check data is included within the rectangular boundaries of the data group, the position of the check data is taken into account if necessary in determining the logical drive position and extent offset address spans of the data blocks. This can be accomplished using information from array control block 720. More particularly, the I/O request server can determine the logical drive position and extent offset of any check data within the boundaries of the data group by examining the type field 760 and the redundancy group stripe depth field 762 of the appropriate redundancy group extent descriptor block 746 (the I/O request server can determine which extent descriptor block 746 is appropriate by finding the extent descriptor block 746 having an extent number field 747 that matches the corresponding extent number field 727 in the data group's descriptor block 914). The I/O request server is directed to array control block 720 by the pointer 718 in the data group mapping element 908.

To translate each logical drive position and extent offset address span to a physical address span on a particular physical drive of the parallel set, the I/O request server reads the physical drive identifier blocks 745 to determine the physical drive corresponding to the identified logical drive position. The I/O request server also reads the base field of the appropriate extent descriptor block 746 of array control block 720 (e.g., application base field 752), which provides the physical address on the drive of the beginning of the extent. Using the extent offset address span previously determined, the I/O request server can then determine for each physical drive the span of physical addresses that corresponds to the identified extent offset address span.

It may occur that during operation of a parallel set one or more of the physical drives is removed or fails, such that the data on the missing or failed drive must be reconstructed on a spare drive. In this circumstance, the configuration of the set must be changed to account for the new drive, as well as to account for temporary set changes that must be implemented for the reconstruction period during which data is regenerated from the missing or failed drive and reconstructed on the spare. It is noted that the configuration utility can be used to remap the set configuration by redefining the parameters of the configuration database.

In general, to those skilled in the art to which this invention relates, many changes in construction and widely differing embodiments and applications of the present invention will suggest themselves without departing from its spirit and scope. For instance, a greater number of second level controllers and first level controllers may be implemented in the system. Further, the structure of the switching circuitry connecting the second level controllers to the disk drives may be altered so that different drives are the primary responsibility of different second level controllers. Thus, the disclosures and descriptions herein are purely illustrative and not intended to be in any sense limiting. The scope of the invention is set forth in the appended claims.

What is claimed is:

1. A system for storing data received from an external source, comprising:

at least two control means for providing control of data flow to and from the external source;

a plurality of storage means coupled to said at least two control means wherein said storage means are divided into groups and each group is controlled by said at least two of said control means such that in the case that a first control means coupled to a particular group of storage means fails, control of said particular group is assumed by a second control means;

a plurality of data handling means coupled to said at least two control means for disassembling data into data blocks to be written across a group of said storage means; and

error detection means coupled to said control means and said storage means for calculating at least one error detection term for each group of storage means based on the data received from the external source using a selected error code and providing said error detection term to be compared with data to detect errors, said error detection means being coupled to each of said control means to receive the data from said control means and transmit said error detection term to an error code storage means in said group of storage means.

2. The system of claim 1 wherein said data handling means further include assembly means for assembling said data blocks received from said control means.

3. The system of claim 1 further comprising a first bus for coupling to said external source and a plurality of buffer means coupled to said first bus and to said control means for buffering data received by and transmitted from the system.

4. The system of claim 3 further comprising error correction means coupled to said error detection means for correcting error in data as said data is transmitted from either said buffer means to said storage means through said data handling means or from said storage means to said buffer means through said data handling means.

5. The system of claim 3 wherein the error detection means uses a Reed-Solomon error code to detect errors in said data received from said buffer means and said storage means.

6. The system of claim 3 wherein said data handling means includes detachment means coupled to said error detection means for detaching from the system storage means and buffer means which transmit erroneous data responsive to receiving said error detection term from said error detection means.

7. The system of claim 1 wherein said plurality of storage means comprises:

a first group of data storage means for storing data from the external source; and

a second group of error check and correction (ECC) storage means for storing ECC data generated by said error detection means.

8. The system of claim 1 wherein each of said plurality of storage means stores data and error check and correction (ECC) data in a predefined pattern.

9. In a system including at least two control means for communicating with an external source and a plurality of storage means wherein at least two of the control

means are c nnected to each of the storage means, a method for storing data received from the external source c mprising the steps of:

receiving data from the external source;

configuring the plurality of storage means into groups wherein each group is initially controlled by at least two of the control means such that in the case that one of the control means fails, the storage means of each group is accessible through another one of the control means;

disassembling data into groups of data blocks to be written to said plurality of storage means;

calculating at least one error detection term from said data using a selected error code;

storing said data blocks in a first of said groups of storage means; and

storing said at least one error detection term in said first of said groups of storage means.

10. The method of claim 9 further comprising the steps of:

retrieving said data blocks from said first of said groups of storage means;

calculating a check error detection term from said data blocks using a selected error code;

retrieving said at least one error detection term from said first of said groups of storage means; and

comparing said check error detection term to said at least one error detection term to determine that said data has not been corrupted.

11. The method of claim 10 further comprising the step of correcting said data if it is determined that said data has been corrupted.

12. The method of claim 10 further comprising the step of assembling said data blocks into a form in which it was received from the external source.

13. The method of claim 9 wherein the step of configuring further comprises the step of setting a plurality of switching means to allow said data blocks to be passed between the control means and the storage means in a predefined pattern.

14. The method of claim 9 further comprising the step of detaching a particular storage means upon which said data was stored if it is determined that said data has been corrupted.

15. A system for storing data received from an external source, comprising:

control means for providing control of data flow to and from the external source;

a plurality of storage means coupled to said control means wherein said storage means are divided into groups;

a plurality of data handling means coupled to said control means for disassembling data with data blocks to be written to said storage means; and

error detection means coupled to said control means for receiving said data blocks in parallel form and detecting errors in each data block substantially simultaneously as said data blocks are written to said storage means.

16. The system of claim 15 further comprising data correction means coupled to said error detection means for correcting corrupted data in response to an error detection signal provided by said error detection means.

17. The system of claim 16 wherein said plurality of storage means comprises:

a first group of data storage means for storing data from the external source; and a second group of error check and correction (ECC) storage means for storing ECC data generated by said error correction means.

18. The system of claim 16 wherein the error detecti n means uses a Reed-Solomon error code to detect errors in said data received from said buffer means and said storage means.

19. The system of claim 16 wherein the error correction means uses a Reed-Solomon error code to correct errors in said data received from said buffer means and said storage means.

20. The system of claim 15 further comprising detachment means coupled to said error detection means for detaching a particular storage means from the system which has provided corrupted data as determined by said error detection means.

21. The system of claim 15 wherein said data handling means further includes assembly means for assembling said data blocks received from said control means.

22. The system of claim 15 further comprising a first bus for coupling to said external source and a plurality of buffer means coupled to said first bus and to said control means for buffering data received by and transmitted from the system.

23. The system of claim 15 wherein each of said plurality of storage means stores data and error check and correction (ECC) data in a predefined pattern.

24. In a system including control means for communicating with an external source and a plurality of storage means, a method for storing data received from the external source comprising the steps of:

receiving data from the external house;

disassembling the data int groups f data blocks to be written to said plurality of storage means;

calculating at least ne error detection term for each data block substantially simultaneously; and

storing said data blocks and said at least one error detection term in a first of said groups of storage means substantially simultaneously.

25. The method of claim 24 further comprising the steps of:

retrieving said data blocks from said first of said groups of storage means;

calculating a check error detection term from said data blocks using a selected error code;

retrieving said at least one error detection term from said first of said groups of storage means; and

comparing said check error detection term to said at least one error detection term to determine that said data has been not corrupted.

26. The method of claim 25 further comprising the step of correcting said data if it is determined that said data has been corrupted.

27. The method of claim 25 further comprising the step of assembling said data blocks into a form in which it was received from the external source.

28. The method of claim 24 wherein the step of configuring further comprises the step of setting a plurality of switching means to allow said data blocks to be passed between the control means and the storage means in a predefined pattern.

29. The method of claim 24 further comprising the step of detaching a particular storage means upon which said data was stored if it is determined that said data has been corrupted.

* * * * *